



Escola de Camins
Escola Tècnica Superior d'Enginyeria de Camins, Canals i Ports
UPC BARCELONATECH

SIMULATION OF A PUBLIC E-BIKE SHARING SYSTEM

Tesina realitzada per:
Víctor Casado Pérez

Dirigit per:
Francesc Soriguera Martí

Intensificació:
Transport i territori

Codi:
722-TES-CA-6849

Barcelona, **22 de maig de 2016**

ITT - Departament d'Infraestructura del
Transport i del Territori

TESINA FINAL DE CARRERA

Abstract

Title: Simulation of a public e-bike sharing system

Author: Víctor Casado Pérez

Tutor: Francesc Soriguera

Urban areas are in need of efficient and sustainable mobility service and it seems that one of the options that arouses more interest are public bicycles systems, which in the last decade have been promoted in cities all over the world almost exponentially, surpassing the 500 bike-sharing programs worldwide. This has resulted in a continuous evolution of these systems, which, due to their recent appearance, have practically gone ahead of the works focused on their study.

This thesis intends to develop an agent-based simulation model to emulate a bike-sharing system, in order to allow the optimization of the main strategic and tactical system variables, such as the total number of bicycles, the number of reposition equipment, the size of the stations and the occupancy after being repositioned, that is, its balanced occupancy. The model will be applied and validated with the parameter values of the system currently applied in Barcelona and an improved scenario will be posed to analyze the performance of a better hypothetical solution for balanced occupancy as an example of the simulator potential.

The simulation is coded in Matlab® environment, using the paradigm of Object Oriented Programming, which aims to ease its use, understanding and expansion, following the open-source philosophy, which has proliferated in recent years.

The document contains the description of the model and the performance of the program, its structure, and its validation. In addition, the source code can be found attached for its free download and use.

Acknowledgment

This study has been made possible thanks to the guidance of Francesc Soriguera, tutor of this thesis, whose interest has achieved to hook me into this work, posing small challenges that have ended up shaping these lines. Thanks for the patience and the thorough supervision.

I also want to thank Marcel for his support in meetings and his small tools, Enrique Jimenez for his parallel work and their particular view of the problem, and of course Anna Llopis in presenting a great base and a better reference.

And not forgetting Claudia, David, Marina, Noga, Laia, Roger, Sergi, Lluç, Dani, Marta, Joan, my enigmatic friends, “ridente” and “Rébol”, Jordi, Víctor... everyone. Your support and patience have been fundamental. I promise to subscribe to the *Bicing* services soon.

Index

1. INTRODUCTION	5
2. STATE OF ART	8
2.1. Bike Sharing Systems.....	8
2.2. Agent-Based Modeling	10
2.3. Modelling bike sharing systems	10
3. MODELLING THE BIKE SHARING SYSTEM	12
3.1. Introduction to formulation of the problem.....	12
3.2. Demand generation.....	13
3.3. Trip Generator.....	14
3.4. Repositioning.....	16
3.4.1. Trucks assignment pool.....	17
3.4.2. Periodic reposition	18
3.4.3. Continuous reposition.....	19
4. CONSTRUCTION OF THE SIMULATOR	20
4.1. Development environment - Matlab®.....	20
4.2. Program structure	21
4.2.1. Introduction to Object Oriented Programing.....	21
4.2.2. Main script and inputs.....	21
4.2.3. Classes	23
4.2.3.1. Main Classes	24
Class <i>City</i>	24
Class <i>Patch</i>	24
Class <i>User</i>	24
Class <i>Bike</i>	24
Class <i>Truck</i>	25
4.2.3.2. Auxiliary classes	25
Class RepoPool	25
Class Statistics	25

Class RepoStatistics	26
4.3. Simulation Procedures	26
4.3.1. Initialize the city	26
4.3.1.1. Set stations.....	27
4.3.2. Time simulation of the service	27
4.3.2.1. Create a user.....	28
4.3.2.2. Move users	28
4.3.2.3. Background procedures	29
4.3.3. Time simulation of the reposition operations	31
4.3.3.1. Continuous repositioning.....	31
4.3.3.2. Periodic reposition	32
4.4. Outputs.....	33
5. APPLICATION AND VALIDATION OF THE SIMULATOR	34
5.1. Input parameters from Barcelona's system	35
5.1.1. Visualization and analysis.....	40
Demand analysis	40
User analysis.....	42
Reposition analysis	46
e-bike analysis.....	51
Costs analysis – KPIs.....	52
5.2. Optimum initial situation: an improved scenario	55
5.2.1. Visualization and analysis.....	56
6. CONCLUSIONS	59
7. FURTHER WORKS	61
8. REFERENCES	64
APPENDIX A1: USER MANUAL	
APPENDIX 2: SUMARY OF CLASS PROPERTIES AND METHODS	

1. INTRODUCTION

In a current society increasingly concerned about sustainability, the bicycles seems to be the most appropriate transportation mode in the urban environment, where the use of automobiles for short distances is increasingly penalized, especially in the downtown cores and poles of demand. Thus, large cities are turning towards efficient and sustainable mobility, avoiding traffic congestion and reducing noise and pollution.

This new paradigm has led worldwide cities to adopt bike sharing systems as part of the urban mobility, extending the accessibility of public transportation systems. The idea is that the users can take a bike whenever they need it and leave it behind when they reach their destinations.

From what we have seen, the system seems simple to implement, but requires a significant investment, so that characterization, sizing and strategic choices are crucial to avoid unnecessary costs. It could be the case of an oversized scheme, which would lead to unnecessary investment, while an undersized system could result in a disproportionate operation cost associated to reposition operations, or in an excessively low level of service, in which case the proposed program would eventually fail.

Likewise, the lack of foresight regarding service demand can lead to failure, as has happened in small cities such as Terrassa or Granollers, near to Barcelona, after joining to what appeared to be in fashion during the last decade, but without studying their feasibility. In conclusion, foresight and thorough study are essential in order to avoid wasting money when designing a bike-sharing system.

With the aim of helping in the statement design, this master thesis intends to develop an agent-based simulation model that emulates a public bicycle system. Analytical tools have been developed to optimize strategic and tactical variable, such as the total number of bicycles, the number of replacement equipment, the size of the stations or its occupancy after being repositioned, that is, its balanced occupancy. However, this need of strong simplifications and the simulator is a much more realistic environment, so it can be used to test optimal designs. Furthermore, the simulator will be useful to assess the operational performance of the system, a level of detail not possible to achieve from analytical models. Alternatively, for given values of these variables (i.e. for current designs) the model will allow evaluating the key performance indicators of the system and its costs.

This simulator will be developed in the environment provided by the Matlab® MathWorks software, using object-oriented programming, as it is widely adopted as the most common paradigm for Agent Based Modeling frameworks because of the similarity between the agent concept and the performance of objects and methods.

Firstly, the model in question is presented specifying in detail all the assumptions stated for its development and contextualizing each procedure in bike-sharing systems, with special attention to the processes of trip generation and repositioning routines.

Then everything related to the created program and its development is exposed. The user's performance is introduced, with explanations of its features and elements that interact with him. This way, it is determined how the program is internally structured and the logical procedure followed to obtain the results. This section is considered of great importance to ensure a reasonable comprehension of the structure to allow possible extensions in further works..

Finally, the simulator is tested with the particular case of Barcelona, whose data is known from previous works (Llopis, A. 2015), in order to detect possible errors and correct the model. In addition, an improved potential scenario is posed, derived from the results obtained in the test, with a better balanced occupancy.

In short, the ultimate goal of this master thesis is to provide a practical, adaptable and extensible tool for agent-based models of bike-sharing systems simulation,

and thus allowing the evaluation of their hypothetical performance. The micro perspective of the simulation approach will add some details of realistic fulfilments. This will be a useful tool for the fine-tuning of tentative implementations.

2. STATE OF ART

2.1. Bike Sharing Systems

Since the appearance of the first bike-sharing system in the sixties, it has evolved through three distinct generations. The first generation, known as White Bikes, was created in Amsterdam in 1965, and consisted of 50 bikes (painted white) spread around the city permanently unlocked, for the public to use freely. However, due to the high number of thefts or damage to the bicycles, the system quickly failed after its launch (DeMaio, P. 2009).

The idea was set aside, and it was not until three decades later that Denmark, namely the small towns of Farsø, Grenå and Nakskov, raised an improved public-bikes system in the early 1990's. Thus, the second generation was born, and came full of improvements: the bicycles were distinguished by color and special design, docking stations were designated where bikes could be locked (DeMaio, P. 2009), (Shaheen, S. and S. Guzman, 2011), and small deposits to unlock bikes were requested to users, a fact that gave its name to the system, known as coin-deposit systems. This system came to Copenhagen, which under the name of *Bycyklen*, opened the first large-scale bike-sharing system in 1995, with bicycles specially designed for this purpose, and which is considered the pioneer of most current systems (Søren B. Jensen, 2000). Nonetheless, despite the improvements, users continue to remain under anonymity, so that thefts are still the main problem.

Just a year later, in 1996, a customer-tracking system software was implemented on the Portsmouth University campus in England, where users were using a magnetic stripe card to identify themselves. Hence, the third generation is based on four principles intended to avoid mistakes learned from previous systems: program bicycles are distinguished through special designs, each system employs docking stations, a user interface is needed for check-ins and check-outs at the stations, and finally, the application of advanced technology that allows users to locate, reserve, and access bicycles (Shaheen, S. and S. Guzman, 2011).

Thereby, bike-sharing systems finally succeeded and spread around the world, to the extent that by April 2013 there were around 535 schemes around the world, made of an estimated fleet of 517,000 bicycles (Shaheen, S. et al. 2015), and by June 2014, public bike-sharing systems were operating in 50 countries, including 712 cities, operating approximately 806,200 bicycles at 37,500 stations (Larsen, J. 2013).

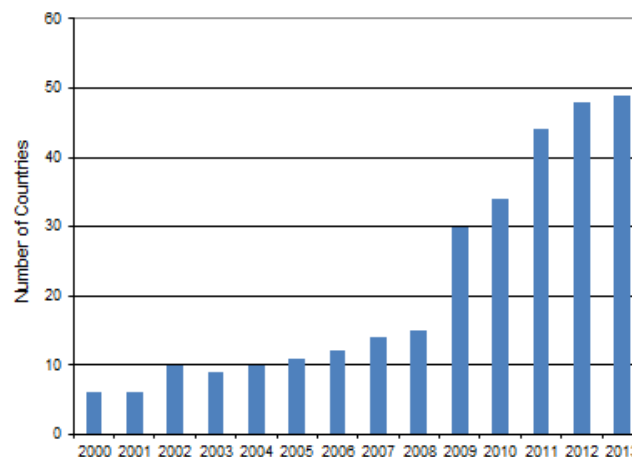


Figure 2.1. Countries with Bike Sharing Systems. (Source: Earth Policy Institute)

Notable programs implemented in Wuhan and Hangzhou, in China, with about 90,000 and 60,000 bikes respectively (ITDP, 2013), are the largest schemes in the world. The one in Paris, called *Velib'*, is the system with greater market penetration, with 1 bike per 97 inhabitants.

Another notable case in the framework of this master thesis is the case of Barcelona, called *Bicing* and launched in 2007, which currently manages a scheme of 6000 conventional bicycles spread over 420 stations, 96.715 subscribers and a ratio of 1 bike per 270 inhabitants. In addition, this program has joined what will be the fourth generation of bike sharing systems, based on the introduction of electric bikes, or e-bikes. This generation will also be characterized by the use of cleaner technologies and incentives that encourage sustainable bicycle redistribution, giving riders a price reduction or additional time credit for leaving bicycles at empty docking stations (Shaheen, S. and S. Guzman, 2011).

2.2. Agent-Based Modeling

An agent-based model (ABM) is one example among a class of computational models for simulating the actions and interactions of autonomous agents, both individual and collective entities, with a view to assessing their effects on the system as a whole (Bonabeau, E. 2002). Thus, ABM is a kind of micro-scale model (Gustafsson, L. and M. Sternad, 2010) that simulates the simultaneous operations and interactions of multiple agents in an attempt to re-create and predict the performance of complex phenomena.

The development of the main idea of agent-based modeling was carried out as a simple concept in late 1940's, when Von Neumann created a theoretical machine capable of reproduction. The idea was put into practice through the called cellular automata, which consist of a collection of cells on a grid built on paper. Years later, in 1970, and with the advent of computers, the mathematician John Conway constructed the Game of Life, Which operated by simply rules in a virtual world (Wikipedia "Agent-Based Model").

Thereafter, during the 70's and the 80's, the first strictly speaking agent-based models appeared. However, due to the fact that further development of ABM required computationally intensive procedures, it did not become generalized until the 1990's, with the appearance of StarLogo in 1990, followed by Swarm and NetLogo in mid-1990's, and AnyLogic in 2000. Since then, modeling software became widely available and the quantity of substances to which the ABM could be applied increased, such as biology, network theory, social science, etc.

Nowadays a big amount of packages that support agent-based simulation can be found, most of them based on Object Oriented principles, as it offers a natural and simple technique for modeling (Allan, R. J. 2010). Thus, agents can be considered to be self-directed objects with the ability of choosing actions based on their environment. Therefore, it seems natural to use class objects and methods to represent agents and agent behaviors.

2.3. Modelling bike sharing systems

Following the fast growth in the interest in bike-sharing as a transportation mode in urban environments, and noted the great potential that this represents for urban mobility in the emerging metropolitan areas in need of efficient and sustainable mobility, there are many who have tried to model it.

The success of the system lies in various aspects such as the location of the stations, which in turn implies the density of stations in the service area and thus

the accessibility of the system; the demand of it, which will characterize the stations (size, occupancy at the beginning of the service, etc.); or repositioning strategies adopted by the operator. The complexity of the problem means that it can be found a lot of studies focused on one of those aspects separately. Specifically it is found extensive literature on the problem of the location of the stations, based on the hub location problem (O'Kelly, M.E. 1986) and other studies derived from it, which try to give a mathematical solution to the design of the stations network of bike-sharing systems (Martinez, L. M. et al, 2012). Some studies go even further and raise the system design strategy from service level constraints (Lin, J-R. and T-H. Yang, 2011). Approaches to this problem applied to reality can also be found, with simulations and optimizations through computational tools (García-Palomares, J. et al, 2012).

However, there are relatively few studies in the literature that focus on the strategic design of the reposition operations, an essential factor in the cost of the scheme, and that is accentuated in cities with unfavorable orography, like Barcelona (Llopis, A. 2015). In fact, it is truly hard to model continuous repositioning strategies, that is to say, running during the service, therefore resort to simulation is almost compulsory.

3. MODELLING THE BIKE SHARING SYSTEM

3.1. Introduction to formulation of the problem

As said before, the objective of this master thesis is to construct an agent-based simulator for an e-bike sharing system to validate the optimal results obtained from the analytical model and simulate the continuous rebalancing.

The basic outline of the performance of the model follows what is generated by a typical mobility problem and the behavior of users against it, using a transportation service, which in this case is a public bike-sharing system. It would be generated from tactical and strategic inputs that define it. The demand is generated across the surface served, from demand data inserted as input from demand analysis.

Once demand is generated, each user located at a point of origin choose a destination, find the nearest station to its current position, walk up to it, pick up a bike, travel to the nearest station to its destination, leave the bicycle and finally reach its destination completing his trip. If some step in this process is not completed, the user will behave as defined below.

On the other hand, the reposition teams basically pick bikes in the stations where leftover and leave them where missing, to ensure the availability of both bicycles and free spots in the maximum as possible number of stations and at all times. This process, which will be explained more extensively in subsequent sections, seeks keeping a good level of accessibility throughout the service, for which is essential to have identified the balance point, that is the occupancy after reposition.

So, before programming, some hypotheses have been developed in order to adjust the simulator to the real problem. That is why some assumptions and work lines are considered and explained below, referring to demand generation in the system, the trip generation for this demand, either destination assignment and trip procedures, and the reposition processes. These issues ultimately intended to set a system based on different models of urban simulation models (EUNOIA, 2012).

3.2. Demand generation

The users are the main characters of the problem: Bike sharing systems exist in order to solve users' mobility needs. Therefore, the problem concerning the demand is a very broad one, which any city that wants to implement a bicycle sharing system, or any other transportation system, must study conducting demand analysis.

However, the objective of this work does not include the estimation of system demand, but focuses on the design and the response of a system considering the demand as a starting point. The same applies to the distribution of stations within the city, which is given by the density of stations resulting from analytical studies to optimize this parameter, such as (García-Palomares, J. et al, 2012) and (Martinez, L. M. et al, 2012). Likewise, features such as size and initial occupancy data are directly related to demand analysis and therefore, out of the scope of this master thesis. In summary, the demand for each station and the location thereof is considered given.

From here on, it is assumed that the demand is generated from the input data according to a Poisson distribution for each station and each simulation time interval. This demand is distributed uniformly over the influence zones of each station, which are determined as Voronoi polygons¹ (also known as Thiessen polygons or Dirichlet tessellation) that bring together all those points whose

¹ **Voronoi polygons:** a Voronoi diagram is a partitioning of a plane into regions based on distance to points in a specific subset of the plane. For each seed there is a corresponding region consisting of all points closer to that seed than to any other (Wikipedia "Voronoi diagram").

distance to the current station is minimal compared to other stations, as it is shown in Figure 3.1.

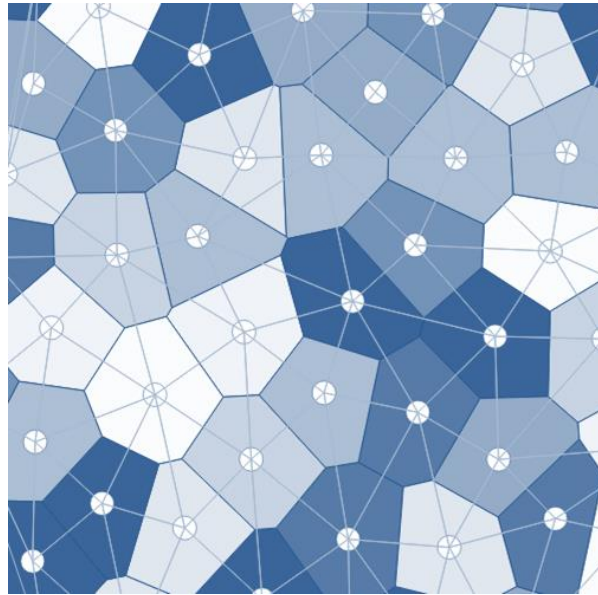


Figure 3.1. Voronoi Polygons

Once a user has an origin, the access time will be computed between there and the station, referring to the Euclidean distance in L2. It is assumed that it exist a maximum distance that the user is disposed to walk to access the system (W_{\max}), so if the access distance computed is greater than W_{\max} the user will not get into it. That would be related with the stations density, so if the latter is low, the number of users that would be interested in the system will be low too. The same would happen with the access at destination.

Another assumption relating to users is that a user's App can be available to report on the status of the system. Thus, the user can know if there are bikes available in the closest station, and indeed on other stations around. This application is also supposed to be available in each station through a display, so the user will know the occupancy of all the stations, also in destination. Hence, it is assumed to that users will not queue at destination station if it is full, they will look for the closest one with free spots to leave the bike.

3.3. Trip Generator

Once a user is generated, a trip must be assigned. As the origin-destination matrices are hard to get, especially for hypothetical or virtual scenarios, it has been designed a trip generator model for the system.

Firstly, it assigns a trip distance (d_s), following a probability distribution. As no empirical data is known, some other distributions have been checked to fit a real situation. Negative exponential distribution seems to be the most effective item in order to represent this situation, but some problems appear while adjusting it: as it only depends on one parameter, the flatness cannot be adjusted and hence, a significant amount of trips result too long. In Gamma distribution it is possible to adjust the shape, but for this case, when calibrating for high values, the distribution for low values distances relevantly from reality. Eventually, a Poisson distribution has been chosen, as is done in other researches (Vogel, P. 2014).

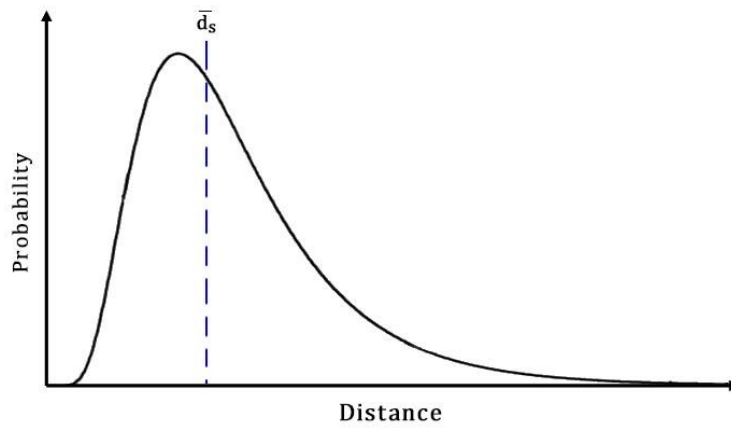


Figure 3.2. Poisson distribution for trip distances

Then, potential destinations are obtained according to:

$$d_s - W_{\max} \leq d_s \leq d_s + W_{\max}$$

To select the final destination station among those potential destinations, weighted probabilities will be calculated by:

$$P_i = \frac{\lambda_t(i)}{\sum \lambda_t}$$

Where λ_t represents the returns rate for each station. For that, it will be assumed that this rate will be set as input of the simulator for each station, as the demand was. Finally, the trip distance between stations will be computed assuming a L2 metric, and also the trip time.

As demand generation does, the final destination will be set considering the influence area of the destination station, assuming uniform distribution of destinations.

3.4. Repositioning

The distribution of trips over the city is rarely uniform. Actually, asymmetric displacements are the main problem regarding accessibility of bike-sharing systems, as it turns into accumulation or lack of bicycles available in the stations, and finally the system collapses.

This phenomenon is basically due to two reasons: the slope experienced by the users during their trips due to the topography of the city –that most of times allows making the trip in downhill direction, but not the trip back uphill– and the decentralization of the demand, which generally causes that users travel only in one and same direction (i.e. downtown, business areas, etc.). (Llopis, A. 2015).

As a response to this asymmetric distribution, a strategic reposition system is widely established, whose function is to assure that bikes are available for almost every potential user. Thus, a system consisting of trucks and operators relocates bikes around the stations, collecting them on saturated areas and leaving them where they are needed following an equilibrium model, that is to say, they compensate the system.

The repositioning operations can be carried out periodically or continuously. Each of them involves different strategies, introduced below, which are deeply studied by research teams all around, pursuing their maximum efficiency. Since it is not the main objective of this master thesis, simply operation systems will be adopted for each one, as an example in our model. Nonetheless, the author is aware that in any case are they the optimum systems; much better strategies can be found. Besides, the procedures of reposition implemented in the simulation will be externalized as public functions to ease the future optimization of the simulation concerning this operation. That is, the assignment of destination for a truck and the calculating number of operation processes will be coded on external functions.

The system will be based on several parameters that will be asked to the user as inputs. These will be: the number of trucks (n_k , m_k) for both periodic and continuous reposition; the capacity of a truck (k), i.e. the number of bikes that it can carry; the average speed of a reposition truck in an urban environment (v_k); and the average time that an operator needs to load or unload one bike and place it in the truck or station (δ). The strategies adopted by the author are described below.

3.4.1. Trucks assignment pool

Before considering further details, with the aim to propose a reposition strategy for both continuous and discontinuous operations, a tool has been created to solve the problem of routing or assigning a destination for all that trucks operating in the system. This tool consists of a Pool binding together all those stations deemed to be visited, either by excess or lack of bikes, and all trucks which are in disposition to search for a next destination. In this case, all those trucks that are stopped –not travelling– have been considered; either those waiting to have an assignment (idle) or those who are in a station operating.

The problem is simple: to assign each truck to a destination station in the most efficient way as possible, namely, the Assignment problem. It is one of the fundamental combinatorial optimization problems, that consists in finding a maximum weight matching (or a minimum weight perfect matching in the case that concerns this study) in a weighted bipartite graph ([Wikipedia “Assignment Problem”](#)). There are many algorithms devised to solve this linear problem within time bounded by a polynomial expression of the number of agents, but the best known, and which will be implemented in this work, will be the Hungarian method ([Larson, R.C. and A.R. Odoni, 1981](#)).

Thus, a minimum pairwise matching will be calculated, and each truck will be assigned to a station so that the resulting set of all assignments will have the minimum distance per operation needed, prior restraint of those combinations that the occupation of the truck will not allow, i.e. If the truck is full it cannot visit stations with excess of bicycles, or vice versa.

Once obtained the assignment, those trucks that are operating in a station will continue with their work without leaving the Pool, and those that turn to be idle will move to the assigned station, in which case they would be removed from the pool, as well as the station.

This tool aims to achieve a better result than that it would be obtained by instantaneous assignment, as each truck takes into account the situation of the other reposition teams when selecting a destination, and thus avoiding unnecessary crosses on their routes.

The whole process is shown schematically in Figure 3.3.

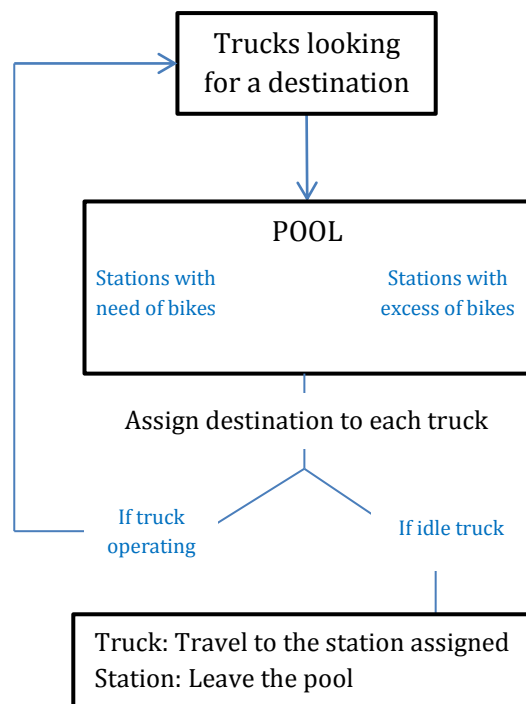


Figure 3.3. Truck assignment pool operation scheme

3.4.2. Periodic reposition

In a periodically redistributed system the operator returns the bicycles to their initial locations² every given period of time, for example, every 24 hours, when the system has a low demand or it is closed. This strategy is many times relaxed so that only a subset of the fleet is redistributed at the end of the service period, so those bikes that would imply a high cost if being relocated (i.e. if a truck has to visit a far station to take or leave a lonely bike) will be unconsidered.

With this purpose, and using the reposition pool launched before, a model is presented in which at the end of the bike-sharing service all those stations that are uncompensated respect to its initial or equilibrium situation are introduced into the Pool. The trucks assigned to this reposition process will access to it as they are refilling stations, so that every time there will be fewer stations available in the pool. When no stations remain in it, the reposition process will be over.

² Each bike is not relocated to its initial location, but the result of repositioning operations results the same as initial distribution.

As has been said before, these processes can be relaxed, avoiding operating in those stations whose visit would imply a high cost. This makes those stations, which are in a very close situation to equilibrium, not enter the Pool at the beginning, and therefore, these bicycles missing in other stations.

3.4.3. Continuous reposition

Continuous redistribution involves relocating the bicycles while the system is in operation. In this case the trucks will take bicycles from stations with excess of bikes and leave them in stations that are close to be out of stock. In practice, continuous reposition offers many advantages over periodic reposition, since the latter requires many employees for short periods of time, while the former can be done with few employees during all day (Barrios, J. A. 2011).

For this matter, some models have been studied, such as a model based on fixed routes, where every truck knows beforehand the route to follow independently of the current state of each station. This way, the reposition truck would follow the tour, and at each stop would load the excess of bikes if it had enough capacity, or unload all needed bikes if available on it, and then it would jump to next stop. However, the best solution has been considered as the one based on the truck assignment pool.

Thereby, an alarms system has been considered, so when a station is close to collapse, either by excess or lack of bikes, this is introduced into the pool to be allocated to a truck. This requires a threshold alarm (a) from which the station would enter in alarm status.

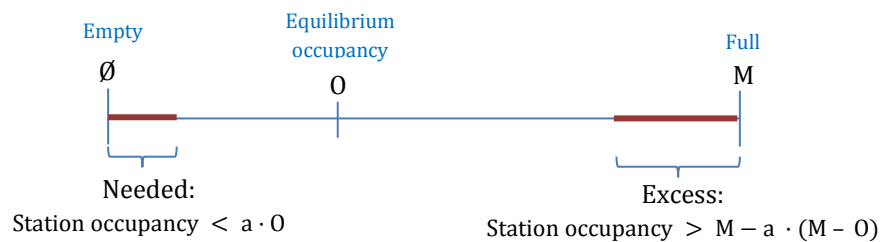


Figure 3.4. Alarms definition scheme for continuous reposition

As trucks depart empty from the central depot, trucks select sequentially to solve the nearest excess alarm from the Pool, that is updated every time step, i.e. every minute. From there on, the assignment process would follow the method explained above in Section 3.4.1.

4. CONSTRUCTION OF THE SIMULATOR

4.1. Development environment - Matlab®

To program the simulator some available software platforms have been evaluated. The first aim was to find a software package that supports ABM simulation, like AnyLogic, EcoLab or Netlogo. The benefits of that kind of package are basically that all of them are focused exclusively on ABM (Allan, R. J. 2010). For that reason, some routines were implemented in NetLogo, but dismissed due to the limitation of its language to generate complex systems and the lack of information about how to implement probability functions.

The second aim was to generate an accessible code, with an extended language that allows keeping on further works on the issue that is concerned. C++ and Matlab languages were assessed as affordable languages.

In the end, Matlab®, from MathWorks, was considered as the best option, as it is a powerful tool for developing technical software, but with a simple interface and language that ease the programmer tasks (Morato, J. 2015). Even more, Matlab lets both object-oriented programming and procedural programming, a useful characteristic that will allow the user to introduce improved functions, for instance

to reprogram the reposition operations without interfering on current agents. Besides, the author of this master thesis is acquainted with this software.

Among Matlab® features are the matrix manipulation, the representation of data and functions, implementation of algorithms, creation of user interfaces (GUI) and communication with programs in other languages and other hardware devices (Wikipedia, “Matlab”).

4.2. Program structure

4.2.1. Introduction to Object Oriented Programming

ABM programming can be done in any language, but Object Oriented Programming (OOP from now on) is the most appropriate and widely adopted language because the idea of an object is similar to the idea of an agent. Besides, the majority of popular ABM frameworks are based on OOP.

OOP applies to software development the standard engineering practice of identifying patterns and defining a classification system describing those patterns, which allows to simplify complex systems, and to reuse efforts by recycling code. Thus, the OOP improves the way to manage software complexity, which becomes truly important when developing large applications. (MathWorks “Programación orientada a objetos en MATLAB”).

When creating software applications, the categories that can be represented include physical objects, such as a bike or a truck, a virtual entity, such as transport administration, or just information. In OOP, these categories are represented as classes, while data elements concerning a particular class are represented as class properties and its operations as class methods

An object is an instance of a class, so when the program runs, the object is created based on its class and behaves in the way is defined by the class.

4.2.2. Main script and inputs

The main script, as the name suggests, is the most important file for the user. It contains a short description about the present software, and is the responsible to initialize the program. It is the only public file, as it calls the rest of functions, methods and classes to develop the simulation, so it should be the only script the user is acquainted with.

The central section of the main script for the user is the inputs section, where all the inputs needed are called and must be introduced. Those necessary variables are specified on Table 4.1, with a summarized description and their current units.

Input parameters	Description	Units
Time		
TotalTime	Total time simulation time.	Min
TimeReDemand	Demand updating period.	Min
City		
Ebike	E-bike mode: it defines the kind of bikes used on the system: 0. Traditional bikes. 1. e-bikes.	-
App	App mode: it defines if it exists and App used by the users to know the current state of the system (stations occupancy): 0. No App service available. 1. App service available.	-
ContRepo	Continuous Reposition mode: it defines the kind of continuous reposition used in the system: 0. No continuous reposition. 1. Based on fixed routes (Peddling). 2. Based on alarms.	-
PeriodRepo	Periodic Reposition mode: it defines the kind of periodic reposition used in the system: 0. No periodic reposition. 1. Reposition at the end of the service.	-
m	Number of bikes in the system.	# of bikes
ContTrucks	Number of trucks in the system responsible for continuous reposition.	# of trucks
PeriodTrucks	Number of trucks in the system responsible for periodic reposition.	# of trucks
Per4Alarm	Lack of bikes or free spots percentage to set reposition alarm.	%
DepoPosition	A vector of position [x, y, z] in UTM data that locate the central depo for trucks.	m
Users		
Wmax	Maximum distance a user is willing to walk to access to the system.	m
WalkSpeed	Walking speed.	Km/h
AvgRideDist	Average riding distance.	m
BikeLoadUnload	Time needed to load/unload a bike on a station by a user.	Min

Bikes		
BikeSpeed	Riding speed.	Km/h
BatteryCharge	Time needed to charge an empty battery.	Min
BatteryConsume	Time needed to consume a full battery while uninterrupted riding.	Min
Trucks		
K	Capacity of the truck, number of bikes it can carry.	# of bikes
TruckSpeed	Truck's cruising speed.	Km/h
TimeLoadUnload	Time needed to load/unload a bike on a station by and operator of the system.	Min
expBikes	Expensive bikes: minimum movements needed to move a truck	-
Unitary costs		
Gam_b	Bike unitary cost	[€/bike·h]
Gam_e	Operations cost	[€/trip]
C_d	Reposition cost	[€/km]
Beta	Value of time	[€/h]
Beta_l	Extra value of lost time	[€/h]
L	Value of life, cost of leaving the system	[€]

Table 4.1. Needed inputs, description and units.

Those are not the only inputs required by the application, as data matrixes must be filed with information about the stations characteristics (location, performance, and initial state), the boundary of the service region, the demand of each station influence area, and the attractions for each one.

For further information about how to introduce and file those input parameters and matrixes, consult the User Manual, on Appendix A1.

4.2.3. Classes

As it is mentioned before, the OOP is structured in different classes, with their properties and methods, which represent mainly the agents, but are also useful to represent the environment, to carry out auxiliary and external procedures and to collect all the data of interest in an organized way.

Therefore, the classes will act as the structure of the simulation, engaged between them to represent the interaction between agents and environment, and actually make decisions that will determine the evolution of the system.

All properties and methods from classes are summarized in Appendix A2.

4.2.3.1. Main Classes

Main classes represent the physical objects and environment of simulation. Thus, these classes are used to create the agents from our ABS.

Class City

The class `City` is the main class of the program. It is the responsible of creating the rest of the objects from other classes, to call their methods, and after all, to implement the time simulation. Hence, it is the base of our system, where all data is recorded and where every procedure happens.

It stores the created objects in classified cells, but also the input values, so it is called by all the methods, avoiding calling the main script variables, and so it is easier to consult the parameters' values used after the simulation.

As the responsible of the simulation, this class provides the most part of the methods, distributed in 7 categories: constructors, initializers of the city, demand methods, reposition methods, data methods, time simulations methods and visualization and analysis methods.

Class Patch

The class `Patch` represents the stations, or rather, the influence areas of each station. Patch objects as a whole sets a grid of stations that appears a mesh, thus, it defines the fixed infrastructure as nodes, and indirectly the urban environment in spatial dimensions.

Class User

The class `User` is the most used and probably the most important class in the program. It defines the users of the system and their behavior.

There are two kinds of methods for Users: functional methods and decision methods.

The functional methods only obey logic procedures needed to run the simulation.

However, the decision methods try to emulate the behavior of a real human, so that is why they are based in statistics data and follow probability distributions, as explained in depth in chapter 3 of this document.

Class Bike

The class `Bike`, together with the class `Truck`, defines the mobile infrastructure, as it is the transport mode of the system and, actually, the issue of this document.

The properties determine the characteristics of what would be a physical bike, both traditional bikes as e-bikes, but also record the accumulated distance travelled.

The methods simulate as well the functional procedures of a bicycle, primarily if e-bikes, as we consider that bikes travel with the user, as a bike doesn't do anything by itself.

Class *Truck*

The class `Truck`, as said before, is the second group of mobile infrastructure, besides representing the reposition area it serves if any, in which case it does not act just as a vehicle but as a control center of the situation in its area of operation..

There are some properties and methods that depend directly on the reposition mode that should be misused if the reposition mode is not the current mode used, so it is important to take care about it while analyzing results.

4.2.3.2. Auxiliary classes

Apart from the main classes explained above that define the core objects of the simulation, other classes that coexist all along are defined. These auxiliary classes aim to make secondary procedures.

Class *RepoPool*

This class is defined with the sole purpose of making all the internal procedures of the reposition pool, used either in continuous reposition or in periodic reposition, and explained in Section 3.4.1.

The `RepoPool` object performance consists basically in identifying all trucks that are looking for a destination and all stations that have to be visited and replenished, and assign a destination to each truck so that the global path is optimal.

It should be noted that this class is defined to suit the reposition procedures conceived by the author of the master thesis, is directly related with the external reposition functions, and would be unused in the future if these procedures are restated.

Class *Statistics*

This class aim, as `RepoStatistics` do, to record most of the significant data to generate final statistics.

`Statistics` objects, in association with `City` object, are responsible of recording all the direct, combined and generated information in a structured way, and save it in an easily accessible cell.

Thereby, data is recorded by methods while running the simulation as properties in an every-minute `Statistics` object.

Class `RepoStatistics`

`RepoStatistics` objects operate the same way as `Statistics` objects, but they only care about reposition procedures. Hence, data recorded by their methods is concerned with properties.

4.3. Simulation Procedures

The simulation process is divided in three sections which in turn are coded as three functions (or better said, as methods) from the `City`'s applications. Those sections are the initialization of the city, the time simulation of the service and the time simulation of reposition operations if periodic simulation is programed, all of them are deeply explained below, based on the hypothesis for the model stated in Section 3.

4.3.1. Initialize the city

Before simulating it is necessary to have what would be the environment ready. That is the aim of the method `initializeCity`, whose tasks are reading all the inputs, introducing them on the system and generating a virtual city according to them.

Firstly it actualizes those properties of the city that depend on the input simple parameters. Secondly, it generates the boundary of the service area and all the stations as `Patch` objects and sets them with the initial conditions, explained in more details below in Section 4.3.1.1. Finally, it generates all mobile infrastructures: bicycles, which are assigned to a station, fulfilling the free spots to reach the initial situation, and with a full battery if they are e-bikes; and trucks, separately for continuous and periodic reposition, that are located at the central depot.

It is important to highlight some aspects about this initialization regarding the reposition service. In the case that continuous reposition based on alarms or

periodic reposition at the end of the simulation are requested, an object from `RepoPool` class is created to act as an assignation pool, as it has been explained in Section 3.4.1.

Furthermore, if peddling continuous reposition is requested, stations would be provided with an ID of a reposition area while being created. That means that it is considered that the city is divided in a certain number of areas (each one assigned to a repositioning truck). As this strategy has been ruled out as part of the dissertation, further details will not be given on this issue. For more information about continuous reposition based on fixed route, check the code of the simulator..

4.3.1.1. Set stations

As it is mentioned in the Section 4.2.3, the stations are defined as `Patch` objects; it means that they are not only characterized as the physical infrastructure, but also as the whole influence area they belong to.

The first step the simulator runs is to assign to the physical station the capacity (number of spots) and the initial disposition of bikes along them.

Secondly it characterizes the influence areas: Voronoi polygons ([Wikipedia, "Voronoi diagram"](#)) limited by the boundary of the service region are calculated and assigned to each station.

To finish, it creates the previously mentioned user creation mesh. This is merely an orthogonal grid of points separated the distance that would consume half of the time steps defined while walking not to change significantly the result. That mesh fulfills each station influence area, and its points will be used as creation point, so the possible points of creation are spared, and therefore the computational cost is substantially relieved. Moreover, it will simplify the routines to create users and a uniform distribution over the territory will be easily obtained.

4.3.2. Time simulation of the service

Once the environment is adequate, the simulation of the service starts. This is the main block of this simulation, as it tries to simulate a real development of what would be a bike sharing-system in a city such as the one described with the inputs. Thus, it comes to emulate the response that virtual users would have facing a mobility necessity in that environment. To achieve it so, methods described in Section 3 are followed in a procedural way. The process described below is schemed on Figure 4.1.

4.3.2.1. Create a user

The foremost action at each time step is to create users in each station influence area, according to their demand.

The program, as said above, creates an orthogonal grid of points inside the area of influence of each station, with a mesh size equal to the distance that a user would walk in half a minute, as this does not significantly change the outcome because of the level of precision being equal to one minute.

The system runs over all stations in each time step, and from the ratios of demand set to that moment and according to a probability assigned by a Poisson distribution (see Section 3.2) users are generated for each station, and instantly configured by following steps:

1. It records the current time as **time of creation**.
2. **Save input parameters** as properties, walking speed and maximum distance to access the system, w_{max} .
3. **Assign an ID**, in order of creation.
4. **Define the origin**: the users are located on a position of the creation mesh of the current station, uniformly and randomly. The access time is calculated, and restrictions of maximum distance are applied.
5. If the `App` mode is active, it **checks** if there are **bikes available**. If there are none, it looks for the nearest station with them, and applies again the distance restrictions.
6. **Define Destination**: Select a destination station following the process explained in Section 3.3. The final destination is assigned analogously to origin, as users destinations are assigned to a position of the creation mesh of the destination station, uniformly and randomly. Riding time and access time at destination are calculated and restrictions of maximum distance are applied.

It may happen that some users are not assigned to a destination because the distance assigned probabilistically does not reach any potential station, in such case they will be replaced by another user object.

4.3.2.2. Move users

Afterwards, the active users are moved, simulating their trip. This means that when a user is created in the system it has a series of behaviors following subroutines for its case, depending on its initial conditions.

As explained above, the first step is finding the origin station, which will generally be the closest station from the creation point. If a user `App` is considered and there are no available bicycles in that station, the closest station with available bicycles is

selected (if it is not further than the maximum walking distance defined, in such case the user would die³). Since all these conditions have been applied while setting the user, during the time that would take to walk to the origin station (access time) the simulator keeps the user in standby (as it represents that it is walking).

Once this time has expired, the location of the user is set as the station location, and it is checked if there are bikes available in the current station (through the station ID). If this is the case, the method `AssignBike` (Users method) assigns a bike to the user: a random one if working with conventional bikes, or the one with the highest battery level if e-bikes. On the other hand, if there are no bikes available, the user repeats the first step, as it has been considered that each station is fitted with a display that works as the App does. In any case, if the accumulated walking distance to access to the system surpasses the maximum defined, the user dies. It has been taken into account that if available e-bikes have a lower battery level that needed to cover the trip (and the trip is uphill in average) the bikes are considered as unavailable and the user discards the trip and dies.

The second step is to calculate the distance and time to the destination station, since that final destination and station are already selected, as discussed in Section 4.3.2.1, but if the user has been derived to another origin station, these parameters are recalculated. The simulator will act alike the access movement, it keeps the user in standby until this time, that represents the riding time, is reached. At that moment, the user would be at the destination station, and it is checked if there is a free spot to leave the bicycle. If not, the user will select the closest station with available free spots, and will repeat the second step, but recording the data as extra trip data, to compile it and evaluate derivation trips.

Finally, the user leaves the bicycle in the station and calculates the time needed to reach its final destination and actualizes all the data related to the trip and defined as properties of users (all specified in Appendix A2). When that happens, the user dies and disappears from the system.

4.3.2.3. Background procedures

Besides, relocating bikes is executed if continuous repositioning is requested. This procedure will be deeply explained later.

To end, data is collected by the auxiliary data classes, described previously, and the environment is prepared for the next time step.

³ Discard the trip and leave the system

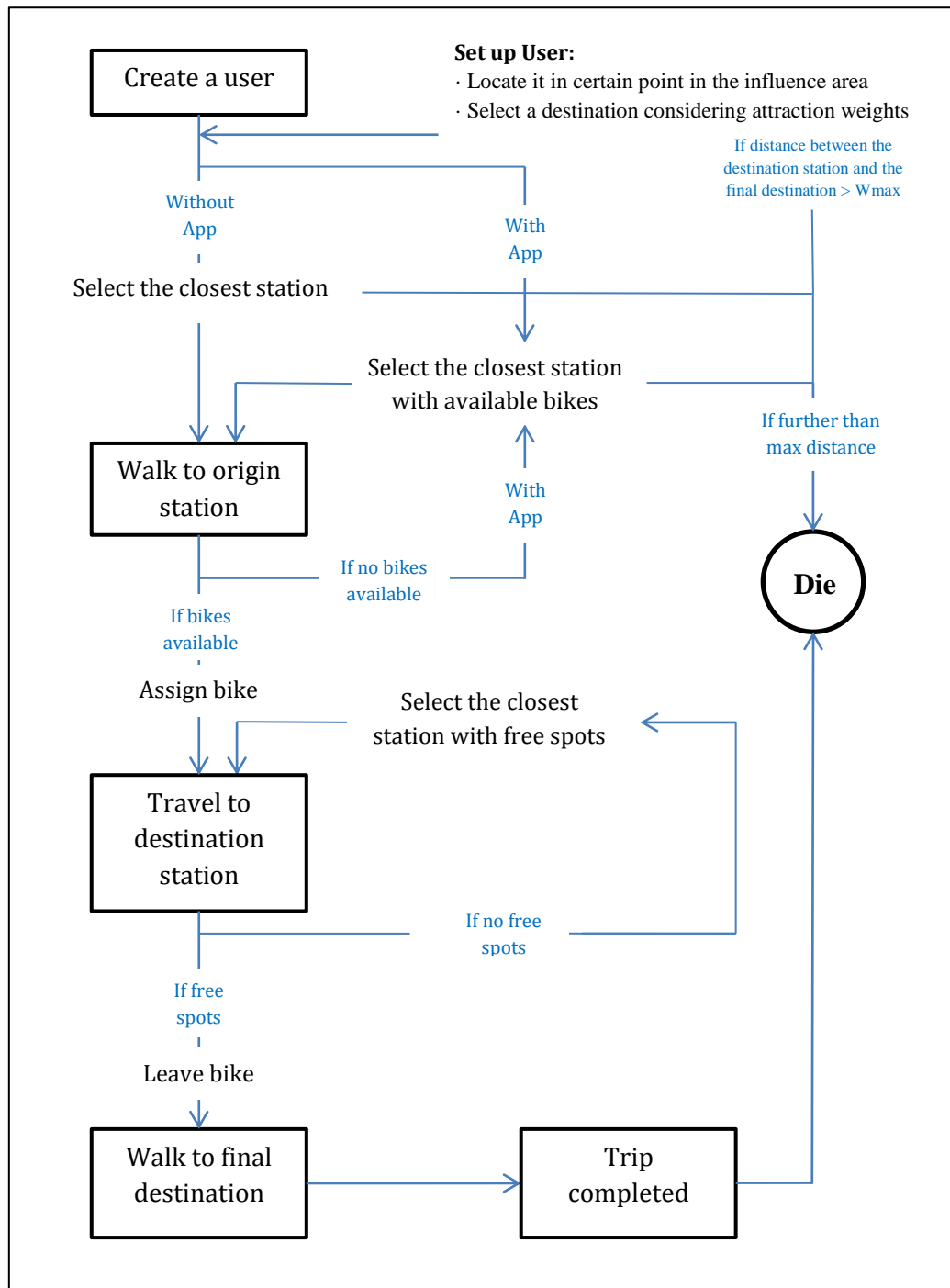


Figure 4.1. Users operation scheme

4.3.3. Time simulation of the reposition operations

To simulate the reposition two strategies have been coded for continuous and periodic reposition. Even so, the starter process is similar for both of them, so if a Reposition mode has been chosen by the user, the trucks are created and set up while initializing the city (Section 4.3.1).

Finally, a system of reposition trucks is settled, with the service region divided in repositioning zones, which are assigned to each truck and all trucks in defined central depot.

These procedures do not involve the rest of the system, including users, as it runs in a parallel way or once the service is finished, depending on the strategy chosen. It is worth remembering that the procedures from now on are not the optimal, and they are externalized as public functions to be replaced by a better solution. Periodic and continuous reposition operations programmed are explained below.

4.3.3.1. Continuous repositioning

This kind of reposition is implemented during the bike-sharing service and therefore, there is not an initial and final situation, since the service is considered not to stop, but an equilibrium position that distorts precisely due to the course of the service. The trucks, therefore, try to compensate these alterations as it is explained in Section 3.4. Thus, those routines will be executed while running the service simulation if the continuous reposition mode has been chosen.

Two routines have been coded and incorporated to the simulator: continuous reposition based on fixed routes, and continuous reposition based on alarms. The first routine has been discarded because the results were significantly worse than the second option but it has been left coded to be replaced at any time by another method, as detailed in the user manual, available in Appendix A1, if future works require it.

Regarding to the reposition based on alarms, the simulator firstly actualizes the alarm level of the stations and calls the reposition Pool procedures in order to define destinations. Then it assigns them to the idle trucks if any, and if there are no stations in need of being rebalanced the trucks remain waiting. It is noteworthy that the trucks do not start travelling until there is a destination station to rebalance. This way they stay in the depot avoiding travelling empty.

Once a destination has been set, the truck is set to “transit state”, and during the time that would take to drive to the station, the simulator keeps the trucks in standby.

When this time has expired, the program sets the truck to “in station” state, defines its new position and calculates the number of operations needed and the time that will take to do so. Then it goes back to standby status.

Finally, it execute the reposition operation procedures, looking for the bikes set on the current station, recording them on the “bikes on the truck” vector, removing them from its location, and actualizing the occupancy of the station (if taking-bikes operations), and the opposite if leaving-bikes operations.

The calculations of the number of operations needed in a station are accomplished by comparing the occupancy of the stations, the equilibrium point assigned to them and the occupancy of the truck. This process is set in an external auxiliary function script, in order to ease its replacement if better strategies are found on further works.

For further information about its performance, consult the functions code.

4.3.3.2. Periodic reposition

The simulation of periodic reposition is started once the simulation of the service is over and if the periodic reposition mode has been chosen. As it is explained in Section 4.3.3.1, this kind of reposition is executed when the system is closed or in a considerably low level of demand. Thus, it starts with an initial situation where the system is as far as possible from the equilibrium situation, and the trucks try to reach it relocating those bikes that are out of their initial position, to allow the system to open another time from its equilibrium position.

For that, a routine has been created in an analogous way to the continuous reposition routine, explained in the previous section, but with the difference that the alarms are updated only at the beginning of the process. As the simulation progress and stations are repositioned they disappear from the Pool. When an idle truck has no destination, it is assigned to return to central depot.

The process finishes in each zone when all the trucks return to the central depot in order to compare the time needed to complete the reposition according to the number of trucks assigned.

4.4. Outputs

The main outputs are basically four matrixes: An instant occupancy matrix, which collects the occupancy rate for each time step and each station; two instant reposition-operations matrixes, one for continuous reposition and another for periodic reposition, which have all the reposition movements completed for each time step and each station as well; an accumulated origin-destination matrix that records all the trips done by the users. Those matrixes will be filed in the Results folder, in the current path, as .xls files, called respectively `Instant_Occupancy.xls`, `Instant_ContinuousRepo_Operations.xls`, `Instant_PeriodicRepo_Operations.xls` and `OD_Acumulated.xls`.

Additionally, the main object from `City` class will be stored in the same folder as `Final_City_Variables.mat`, and all the data recorded on `Statistics` and `RepoStatistics` objects will be available at the end of the simulation, as properties of the `City` object called `vStatistics` or `vRepoStatistics` (containing every step significant information, for time simulation of the service and for reposition service), and `FinalData` or `FinalDataReport` (containing the final situation and some parameter averages reached during the simulation of the service and the time simulation of the reposition operations). These properties are specified in Appendix 2.

From all these outputs, some visualization figures has been programed as `City` methods, that are accessible from the `City` script, and that will be shown in the section 5 in order to validate the simulator.

5. APPLICATION AND VALIDATION OF THE SIMULATOR

In this section, the simulator described in section 4 will be applied in two different scenarios. The first one is the current situation for Barcelona's system, with parameters obtained from the current bike-sharing system in Barcelona, that are described in 5.1, in order to validate the simulator performance and to obtain a referent scenario. The second one, an improved scenario with a better distribution of stations occupancy for the initial and equilibrium situation, with parameters obtained from an optimization performed in other studies (Jiménez, E. 2016).

Firstly, the simulator will be directly run several times with the numerical value of the parameters. This will gave to the author an amount of data, detailed in section 4.4, that will be analyzed and compared to the real situation.

Then, the validation of the simulator will be discussed, and those results that are far from reality will be pointed out to locate where the unexpected results come from.

Finally, further works to improve and develop new applications will be recommended and detailed in section 7.

5.1. Input parameters from Barcelona's system

The Barcelona's system, called *Bicing*, is one of the biggest bike sharing systems in Europe. With 420 stations and more than 6000 bicycles, it serves on almost 100km² ([Bicing website](#)). Some of these stations work as cluster stations. This means they have been aggregated to simplify the performance of the simulator. Besides, as the service region is not defined, it has been considered as a boundary located 1 kilometer away from the border stations. The resultant layout is shown in Figure 5.1 with all the stations located and the service region considered.

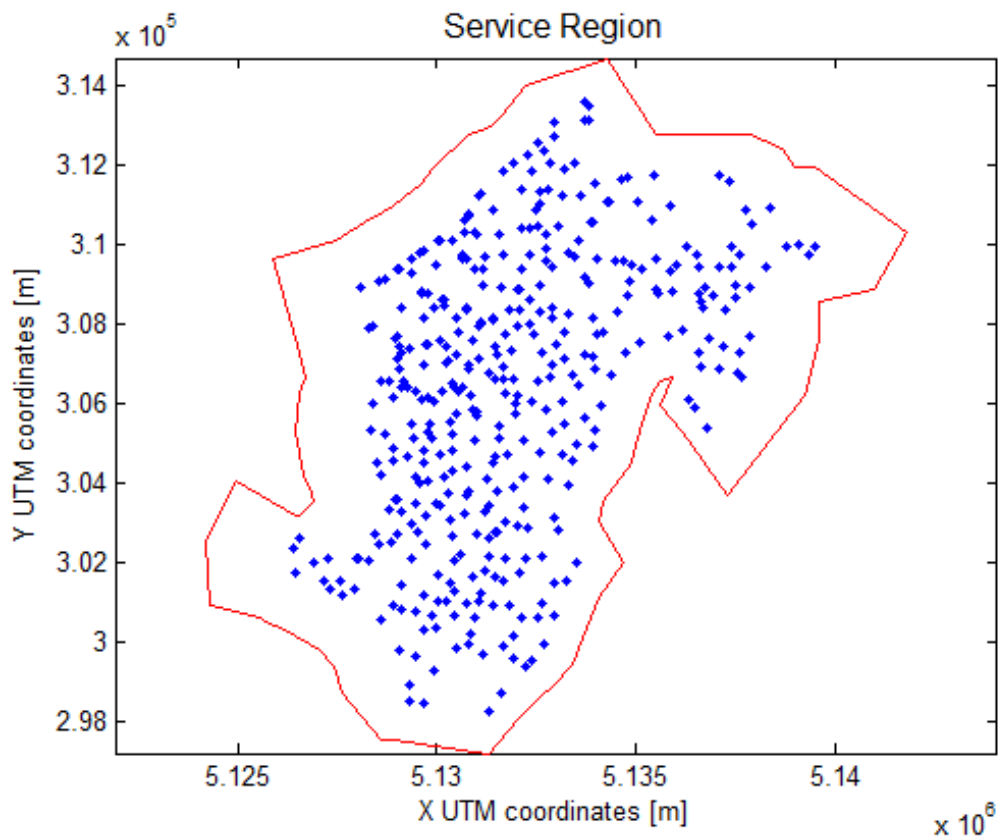


Figure 5.1. Layout

The stations size varies depending on their location, intending to provide enough slots to ensure a good performance of the system. Although the simulation results will confirm that they are undersized and oversized in many cases, the current sizes are shown in the histogram of Figure 5.2.

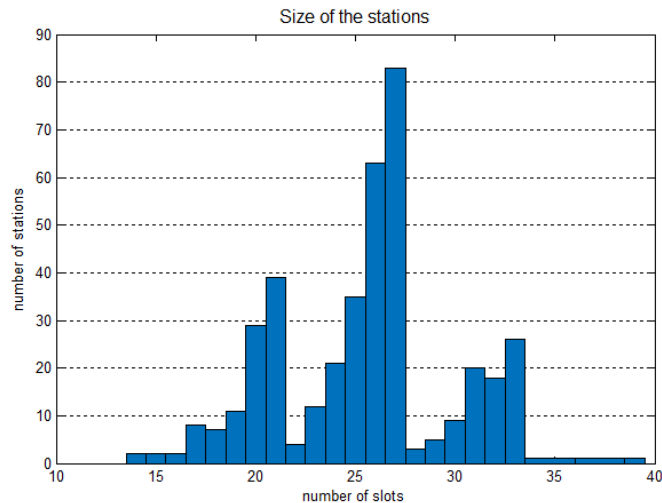


Figure 5.2. Size of the stations

In spite of its particular topography, with a quite constant slope going from the coast up to the hill, this system accumulates more than 1 million uses per month. However, this change in height between stations in barely 6 kilometers implies an added difficulty to users, which may suggest that an e-bike system would have a great performance, as it is being implemented. The cumulative distribution of Bicing stations with height is shown in Figure 5.3, where it is possible to notice that, as most of stations are lower than 110 meters height, it would represent an average slope of 2%.

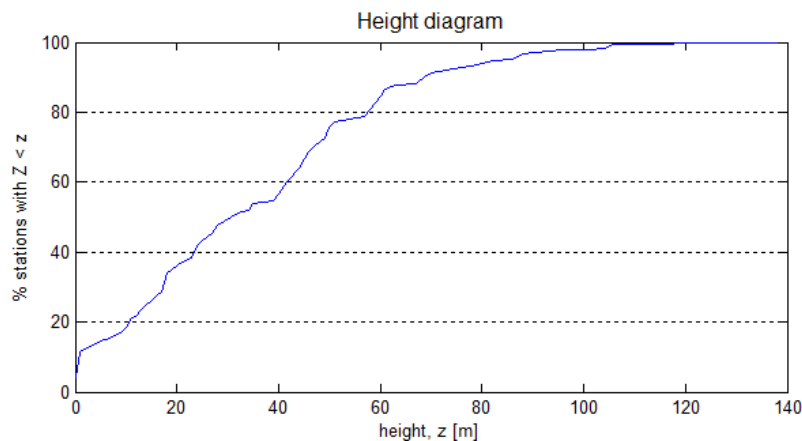


Figure 5.3. Height diagram

Regarding to all the parameters needed as inputs, most of them has been obtained from the information provided by *Bicing* and its site ([Bicing website](#)) and from a previous research ([Llopis, A. 2015](#)). Those parameters are gathered in the Table 5.1.

Parameter	Notation	Units	Value
User parameters			
Average walking speed	WalkSpeed	Km/h	3.6
Average ride distance	AvgRideDist	m	3930
Average time to load or unload a bike to the station by users	BikeLoadUnload	Min	1
Average cycling speed	BikeSpeed	Km/h	15
Bikes parameters			
Number of bikes	m	# bikes	6000
Time to fully charge the e-bike battery from empty	BatteryCharge	Min	120
Time to consume a full e-bike battery while cycling	BateryConsume	Min	200
Reposition parameters			
Number of trucks in the system responsible for continuous reposition.	ContTrucks	# trucks	13
Number of trucks in the system responsible for periodic reposition	PeriodTrucks	# trucks	13
Trucks capacity	K	# bikes	32
Average trucks cruising speed	TruckSpeed	Km/h	21
Average time to load or unload a bike to the station by truck operators	TimeLoadUnload	Min	0.625
Unitary costs			
Bike unitary cost	Gam_b	€/bike·h	0.05
Operations cost	Gam_e	€/trip	0.62
Reposition cost	C_d	€/km	1.78
Value of time	Beta	€/h	11.4

Table 5.1. Bicing inputs

Furthermore, other parameters has been proposed by the author and optimized from the results obtained with the simulation. Those parameters are shown in the Table 5.2.

Parameter	Notation	Units	Value
Position of the central Depot for trucks	DepoPosition	[x, y, z] UTM	[5.1289e+06, 3.0487e+05, 35]
Lack of bikes or free spots percentage to set reposition alarm	Per4Alarm	%	20
Maximum distance a user is willing to walk to access to the system	Wmax	m	750
Expensive bikes (minimum movements needed to send a truck)	expBikes	# bikes	2
Extra value of lost time	Beta_l	€/h	9.12
Value of life (cost of leaving the system)	L	€	9.84

Table 5.2. Other inputs

A particular case is the position of the central depot that has been estimated by the coordinates given by Google Maps for an approximated position of the depot in Barcelona. Likewise, the extra value of lost time, that has been calculated considering that users perceive the lost time 1.8 times more expensive than regular time (Llopis, A. 2015). In addition, the “value of life”, or cost of leaving the system, that in fact measures the cost of transportation alternatives, since if the user has no bikes available it would use another transportation mode. In order to obtain an approximation, the average trip cost in a taxi has been chosen, but it is recommended to estimate a public transportation option for future works, although that was not the object of this study.

Finally, remaining inputs have been decided by the author to generate the most significant situation as possible, trying to adjust the virtual system to the real system used by *Bicing*. These simulation scenarios refer to those called “modes” in the simulation:

The E-bike mode has been activated to generate an approximation of the performance of the batteries, particularly regarding to their level of service and their available usage time, despite they are not totally implemented on the system of Barcelona. In fact, the e-bike system works in parallel to the system considered, with their own network of stations that has not been considered for this simulation. Even so, it would be appropriate to consider this case in order to assess its operation if it is to be implemented in full. However, the results would not be

accurate, as the simulator is considering that cost, performance of e-bikes and users behavior is equal that with traditional bikes, but in reality this might not be the case.

Concerning reposition modes, it is quite complex to simulate the routines that the *Bicing* system carries out, and it is not the aim of this research. Although the routines coded to these works, as it is said in previous sections, are not the most efficient, it seems interesting to analyze the reposition based on alarms, regarding to continuous reposition, and the periodic reposition at the end of the simulation, that is programed following the same assumptions of the previous mentioned, considering alarm situations. These processes are longer explained in section 4.3.3.

These assumptions are summarized in Table 5.3, and they complete all the inputs required by the simulator.

Mode parameter	Notation	Simulation scenarios	Input Code
E-bikes mode	Ebike	0. Conventional bikes 1. E-bikes	1
Continuous reposition mode	ContRepo	0. none 1. based on fixed routes (discarded) 2. Based on alarms	2
Periodic reposition mode	PeriodRepo	0. none 1. at the end of the service	1
App mode	App	0. not available App 1. available App	1

Table 5.3. Mode inputs

The code will be run for a period of 24 hours, or what is the same, 1440 minutes, and considering a period of actualization of demand and attraction data of 60 minutes, attending to the data obtained from *Bicing's* system. This data has been obtained from a web service provided by Clear Channel and recorded for previous works (Llopis, A. 2015). This website is updated every minute with information of the system: number of bicycles and spots in each stations, and precise information on the location and the status of the stations. Thus, the inputs are demand and attraction for the day May 7, 2014, and corresponding to 24 hours after closing time the same day at 2 am.

5.1.1. Visualization and analysis

In order to analyze an appropriate behavior of the simulator, some graphics have been programmed. They attempt to represent some characteristic parameters of the system, so that all processes performed, or rather simulated, can be compared with the real situation of Barcelona's system. This real state has been previously analyzed in [Llopis, A. \(2015\)](#), and the author of this master thesis will base the conclusions on the comparison between the outputs of the simulator, resulting from running the full code several times and processing their significant average values, and the reality.

Below are shown and annotated the visualizations of these significant parameters, divided into categories, depending on the processes they affect to or where they come from.

Demand analysis

The demand is not the same for each run, as it is determined, at each step (every minute) and each station, by a probability function following a Poisson distribution with a sample mean equal to the current demand assigned to each station. By the way, the final demand simulated must be closed to the expected demand introduced as an input, as can be deduced from Figure 5.4.

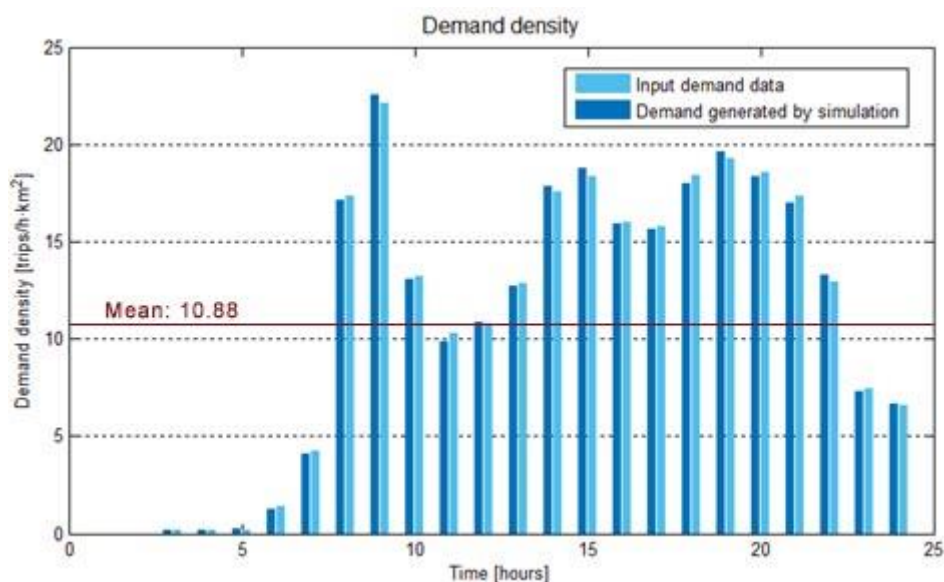


Figure 5.4. Demand density

This demand is just the number of users created by the simulator, which are in total 41980 in average, but it does not consider those users that died because of destination assignation and adjustment problems. For example, as the destination is assigned following a distance probability distribution, some users would be

assigned a travelling distance that would be too large to find a destination station, so it would be defined out of the service region. These mismatches depend on not considered parameters, like the slenderness of the service region but, in the case that concerns us, these dead users represent barely a 3.1% of the total users, and they are actually replaced by another user, which is computed just after death not to alter the results regarding inputs. To sum up, the total number of users created and recorded by the simulator is a few above the real users, but some of them are discarded and replaced by another user in order to adjust the system.

It is possible now to obtain the average demand for different periods of time, and specially the average demand for small time periods, that will result the average of peak demand. In this case, the minute with the highest demand is found, then the two consecutive minutes with the highest average, and so on until considering the 24 hours average demand (1440 minutes), which may result in the lowest average defining a decreasing curve by definition, which is shown in Figure 5.5.

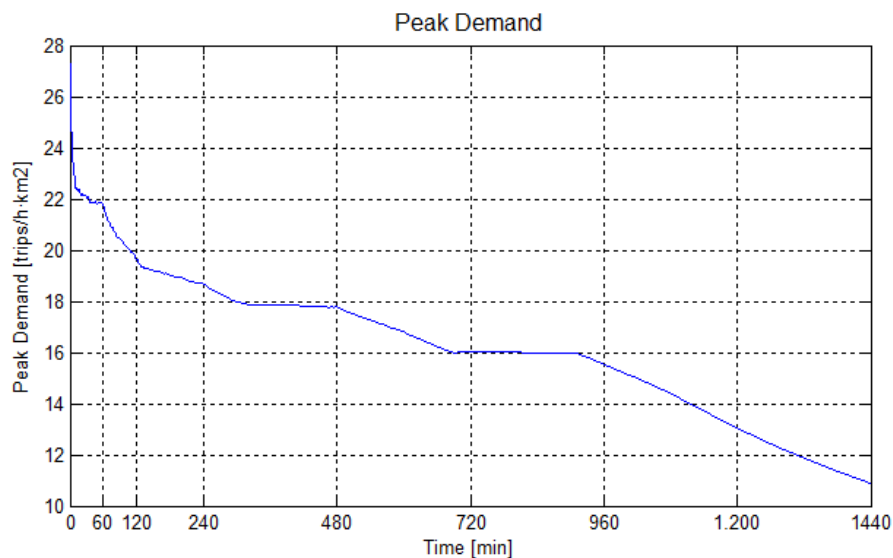


Figure 5.5. Peak demand function for one day

Once the curve defined it is possible to know the average demand that corresponds to a certain period of time. In this case, the average peak demand that is interesting and useful is for a period of 15 minutes, not far from the average travelling time in the system. This peak demand has a value of 22.32 trips/h·km², as it is shown in Figure 5.6, which represents a detail of the curve for the first 2 hours...

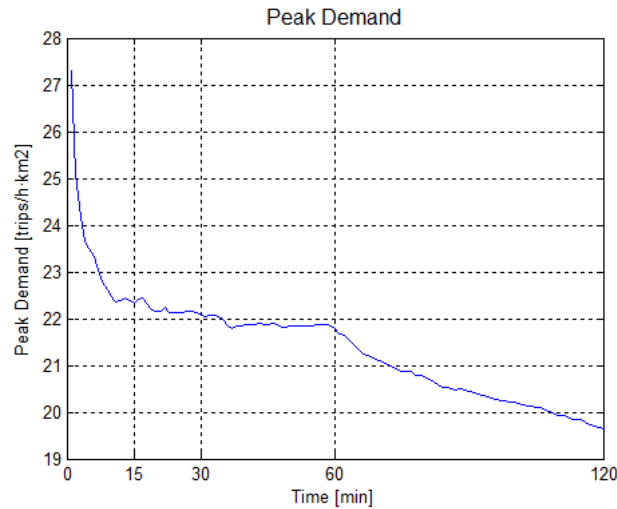


Figure 5.6. Detail of the peak demand function for the firsts two hours

User analysis

The user's behavior is basically influenced by the station density of the system, and therefore, by the distance to the nearest station that implies an access time. Once they take a bike, the rest of the trip depends on them and therefore, it is the operator's responsibility to assure low access times that will guarantee good level of service.

From the results of the simulation, access times are computed on both origin and destination, and despite the lack of experimental data, the values obtained seem to be quite representative of what a real behavior would be. These results have been treated to generate a normalized histogram shown in Figure 5.7.

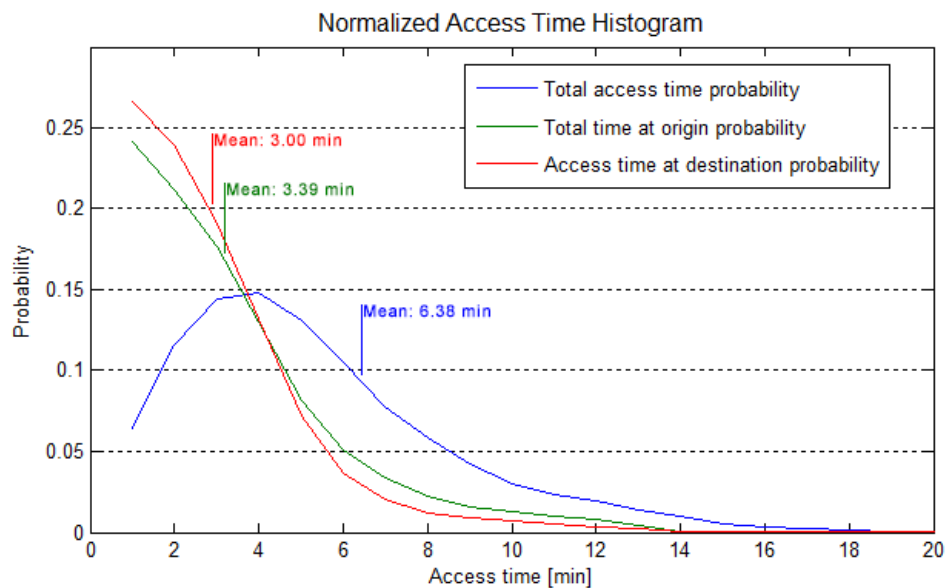


Figure 5.7. Normalized access time histogram

In this case, the access time is determined by the area of the influence zone of each station, as the demand is created uniformly throughout that area, and it will increase when users have problems to find an available bicycle in the closest station, or conversely, when the destination station is full. In such case, users would look for the nearest station with available bicycles or free spots respectively, increasing the distance from their origin to their destinations. As time runs, this situation becomes more common depending on the situation of the stations, as it is shown in Figure 5.8. Moreover, the percentage of empty stations clearly above the percentage of full stations (fact that will be discussed later on), explains the flatness distributed histogram for access time at origin in the Figure 5.7, that result in a higher average than the access time at destination.

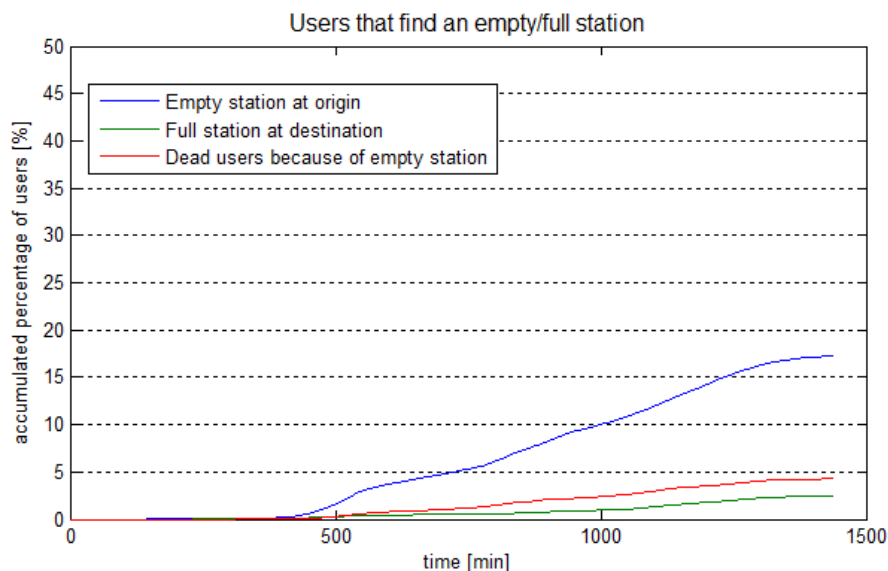


Figure 5.8. Accumulated percentage of users that find an empty or full station

The effect of full and empty stations goes beyond the access time, since there is a certain percentage of users who cannot access directly to the system. This is because in an area within the service region there are not bicycles available, so the access cost is increased very sharply; the user cannot afford it and leaves the system, or as it is said in programming, it dies. In the present simulation an average of 2406 users died due to lack of bikes at origin, which represent 4.3% of the total demand. Nevertheless, those results depend directly on the reposition strategy and the number of reposition teams, and this percentage would be even higher if continuous reposition was not programmed, so the system would become collapsed, as it is shown later in Figure 4.12.

The users' riding time does not depend on the system parameters, but it results directly from the trip generation model chosen. In this case, as a probabilistic model based on a negative exponential distribution in the trip distance, the final distribution of the riding times follows that scheme, as Figure 5.9 shows in a normalized histogram. However, the average distance introduced as an input was 3930 meters, while the output average resulted in 4120 meters, near to 5% over. Thus, this distance is translated in a 23.8 minutes riding time average, versus 22.7 minutes considered as input. The increase should be explained as a consequence of station derivation at the destination due to a lack of free spots to leave the bike, as was considered as well in the access time analysis.

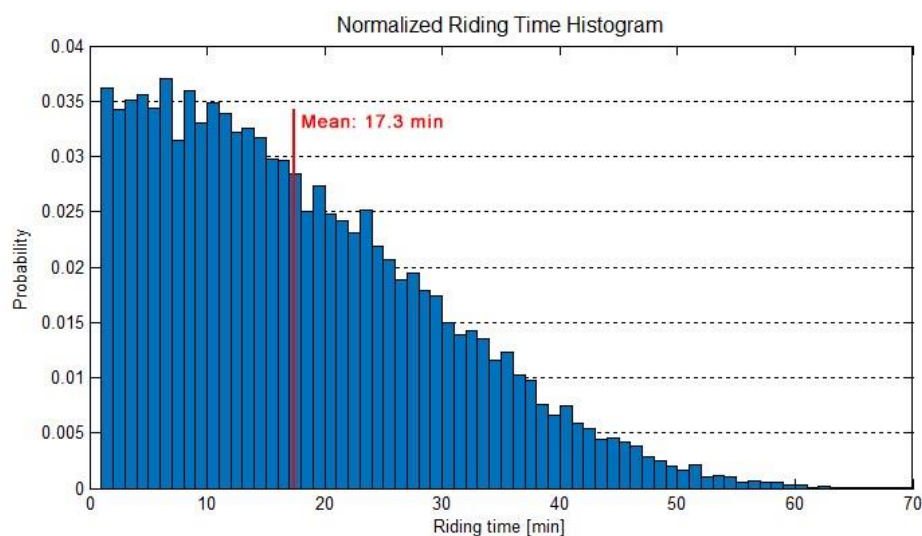


Figure 5.9. Normalized riding time histogram

Nonetheless, some appreciations must be done on these results; as the results follow a probabilistic Poisson distribution, a considerable number of users are expected to travel more than 30 minutes and some of them where assigned to travel even more than an hour. This does not coincide with reality and therefore, a penalty for users who take a bike longer than half an hour is applied, so that from there the demand plummets. In addition, it is hardly credible that users would use the bike-sharing system for long-distance trips, as there are other public transportation systems, such as bus or subway, much more efficient. In any case, the absence of experimental data to determine a distribution that matches closer with the real behavior of users, it is considered that the data obtained through this assumption may result rather tight in average.

To end the analysis of the data generated by the performance of users, cumulative curves about requests and returns over the period simulated are shown. In order to better appreciate the difference between both, as this is much less than the total

volume of operations and plotting directly would not provide any visual information, they are shown as oblique N-t curves (Cassidy, M. J. and M. Mauch 2001). To do so, the theoretic straight line with slope equal to the average ratio of demand throughout the day has been subtracted to the original curves:

$$N'_q(t) = N_q(t) - b_0 t$$

$$N'_t(t) = N_t(t) - b_0 t$$

$$b_0 \approx \bar{\lambda} \cdot T$$

The resultant curve is shown below in Figure 5.10.

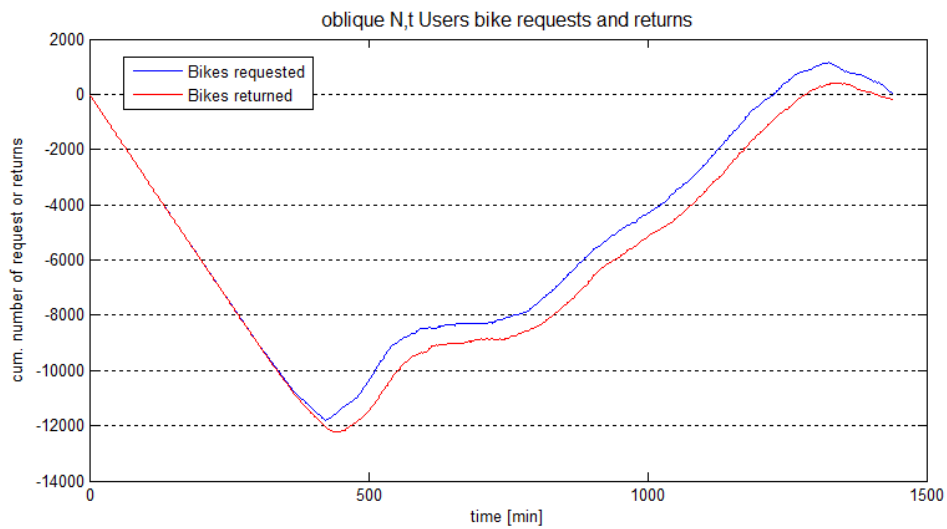


Figure 5.10. Oblique N, t Users bike requests and returns

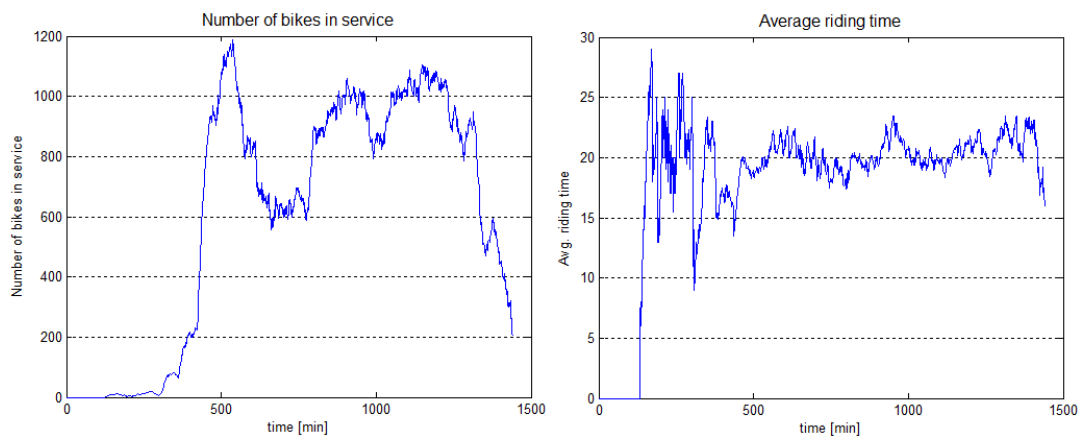


Figure 5.11. Number of bikes in service and average riding at each time step

Computing the distance between original cumulative curves, both vertically (request and returns accumulated) and horizontally (time), the number of bikes in service and the average riding time respectively for each minute is found, as shown in Figure 5.11. At first, it seems the riding time found should be the riding time of the current customer but that is not entirely true because it is not a FIFO (First In-First Out) service, what is obtained is the average riding time of customers who requested the bicycle at each time.

Reposition analysis

Reposition operations, as discussed before, are scheduled so that there is a continuous reposition service running throughout the day, and a periodic reposition service that starts at the end of the public service. The level of service is the most valuable item for continuous operations, keeping the percentage of stations in full or empty conditions as low as possible, and the speed of operation in the case of periodic service, to reach the initial conditions at the beginning of the next service. That is why the results will be analyzed separately, paying attention to different parameters.

Referring to **continuous reposition**, the first point to be analyzed must be the percentage of stations that are full or empty at each step. With this objective, in Figure 5.12 the percentage of simultaneously full or empty stations has been shown in time series, comparing the situation considering or not continuous reposition service respectively.

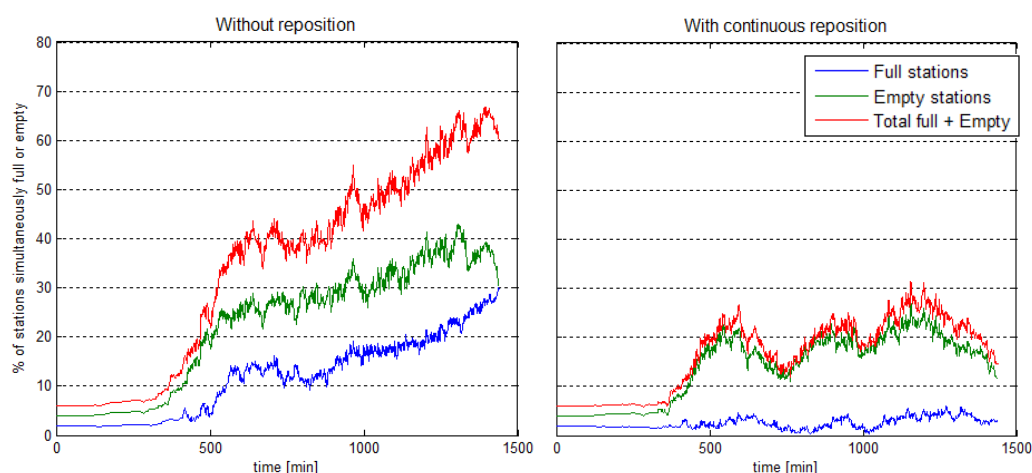


Figure 5.12. Comparison between full and empty stations with and without continuous reposition

The first obvious observation is that while a level of service is kept below a 30% of the stations full or empty with continuous reposition, the system collapses without it, overcoming a 60% of almost useless stations.

The second observation shows that the highest percentage consists of the empty stations, which suggests that, in general, the initial configuration of stations occupancy, which should be in equilibrium solution, may be below the required, conclusion also reached in previous works (Llopis, A. 2015).

Another conclusion, more recognizable in Figure 5.13, where the previous figure considered with continuous reposition has been expanded, is that the shape of the curve reminds of the demand's shape. This fact can be explained by the inability of the trucks to reach the level of requests and returns that users do, so when the demand is higher enough, the stations are filled or emptied above the rate of replenishment of trucks. Conversely, when the demand is low, trucks surpass user movements and balance the system.

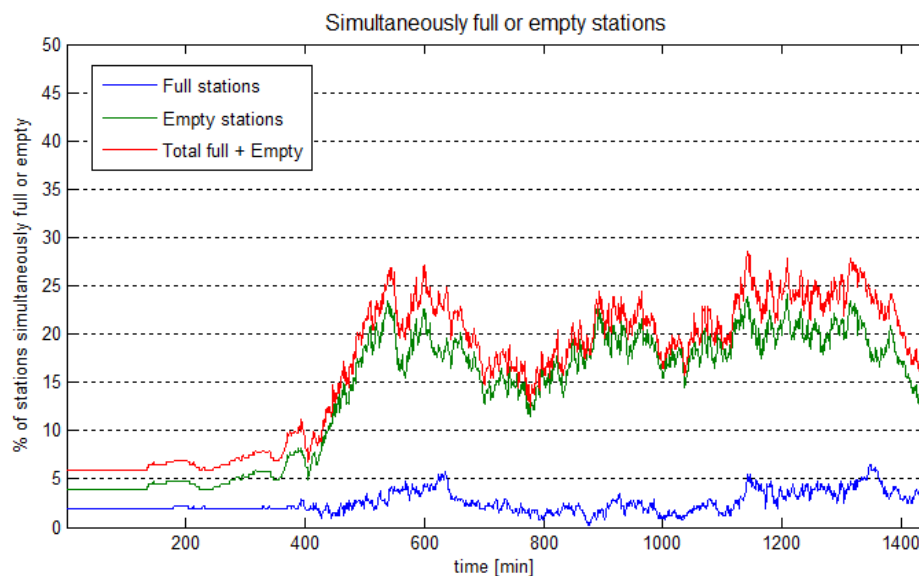


Figure 5.13. Simultaneously full or empty stations

To accomplish this result, the trucks have to visit those stations that are in alarm state (see section 3.4.1). In Figure 5.14 the cumulative percentage of stations visited a specific number of times appears, from stations visited at least one time until a lonely station visited 16 times along simulation.

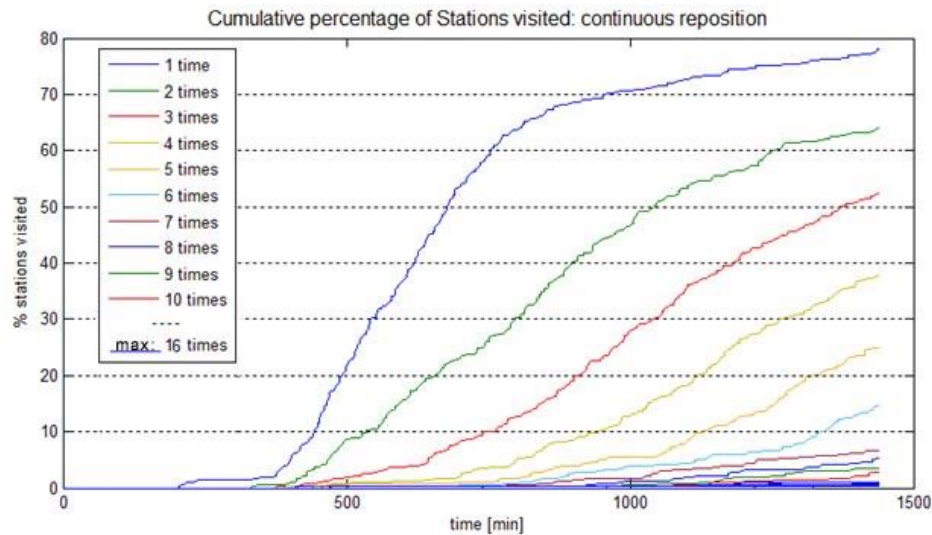


Figure 5.14. Cumulative percentage of stations visited in continuous reposition

It is possible to glimpse that almost 80% of the stations have been visited once, and over 50% have been visited up to 3 times. It seems relevant too that the 8% of the stations have been visited more than 7 times, that could mean that they are in great demand, but also that their level of alarm is quite close to its assigned equilibrium point, as it was commented while analyzing stations occupancy.

As did with users requests, cumulative curves for trucks about requests and returns of the reposition operations over the period simulated can be computed and drawn as oblique N-t curves, in Figure 5.15.

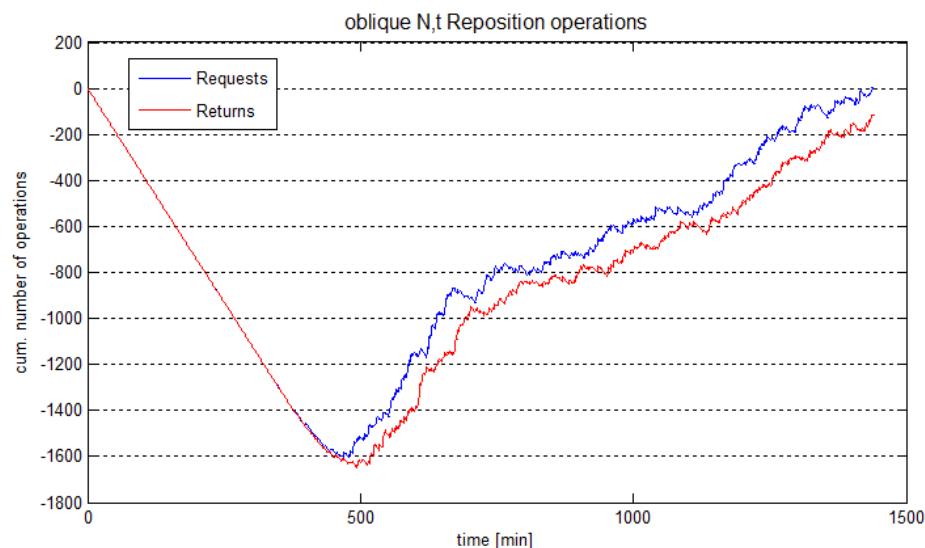


Figure 5.15. Oblique N, t reposition operations

It is important to note that the curves do not close at the end of the simulation. This is because the trucks are not scheduled to leave bikes when the end of the day arrives, so when the service simulation ends about 150 bikes remain on trucks.

This result does not affect in any case the analysis that can be done in studies of bike-sharing systems, but should be considered if consecutive simulations are to be done, so the number of bikes on trucks must be relocated at the start of the next day or during periodic repositioning operations if programmed.

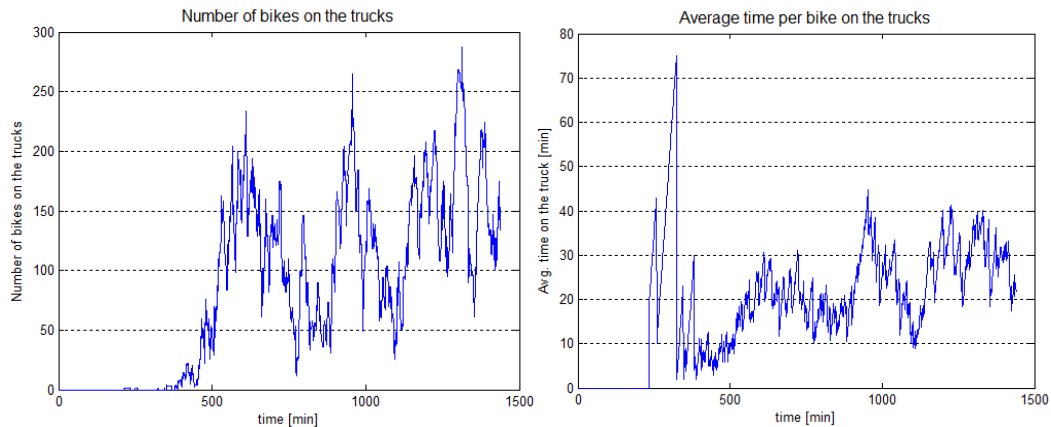


Figure 5.16. Number of bikes on the truck and average time per bike on the truck at each time step

In this case, the distance between the curves vertically represents the number of bikes on the trucks for each value of t , and the horizontal distance represents the average time per bike on trucks, shown in Figure 5.16. It is noteworthy that while the number of bikes on trucks seems to follow no pattern, being an apparently random variable from a certain level of demand, the average time per bike function starts with a high peak (as low levels of demand means that trucks wait for alarms), and then its shape has an increasing trend, but very diffuse.

As a final point, the data of total and average invested time, and total and average distance traveled by replacement equipment are collected. Those are:

Total distance travelled = **1 834 km**

Total time invested = **13 084 minutes**

Avg. distance travelled/truck = **141 km**

Avg. time invested/truck = **1006.5 minutes**

Referring to **Periodic reposition**, one of the most important objectives to be achieved is to finalize the replacement operations throughout the period when the service is stopped or in a very low level of demand. With the aim of determining the time these operations take, as already mentioned, the periodic reposition programed as untimed simulation, so that for each initial conditions and each number of trucks assigned to this task the total time needed will be found, in order to discretize the acceptable values, especially for trucks number.

In the case studied, the maximum time destined to periodic reposition would be a maximum of 5 hours, more than enough time to perform the tasks with the 13 trucks that are available in *Bicing's* system.

Thus, in the system generated for this study, the simulator calculates an estimated time of about 170 minutes to perform all necessary operations, by repositioning routines designed by the author and explained in detail in previous sections.

We also studied the minimum number of trucks that would meet the time constraint, so that the desired situation would be accomplished in less than 5 hours. The results obtained using the simulator determines that the minimum would be 5 trucks, although in some cases it exceeds the deadline. From 6 trucks the condition is always met.

In order to achieve this, the trucks visit each station to reach the equilibrium point of each, excepting those that are close to that point (avoiding expensive trips per bike). The same way as for continuous reposition, the cumulative percentage stations visited a certain number of times during this period is represented in Figure 5.17.

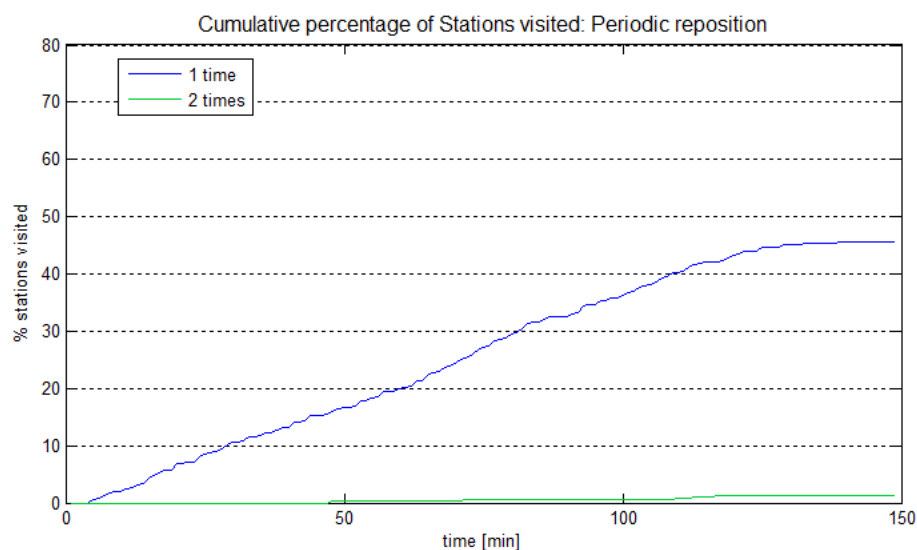


Figure 5.17. Cumulative percentage of stations visited in periodic reposition

What is most remarkable is that less than 50% of stations have needed to be visited, which indicates that the rest are close to that point of equilibrium. In addition, only 2.7% of the stations have been accessed 2 times (11 stations in this case), and none has reached 3 visits.

To conclude this section, the data of total and average invested time, and total and average distance traveled by replacement equipment are collected as for continuous reposition.

Total distance travelled = **261 km**

Total time invested = **1755 minutes**

Avg. distance travelled/truck = **20.1 km**

Avg. time invested/truck = **135 minutes**

E-bike analysis

Due to the lack of demand in studies and patterns of decision regarding e-bikes, the only aspect that has been analyzed in detail for these agents, concerning the technical part thereof, has been the level of batteries available in each moment.

Far from being a restrictive parameter, the low percentage of simultaneous use of the fleet and the short riding periods make all the batteries remain practically at their highest level for almost the entire simulation, and only at peak times distinguish a small disturbance in the global level. To observe it, it is shown in Figure 5.18, which generally shows that the impact is minimal. When zooming in, it is possible to see as predictably, that this disturbance follows the shape of demand diagram, as the batteries only consume while being ridden by users.

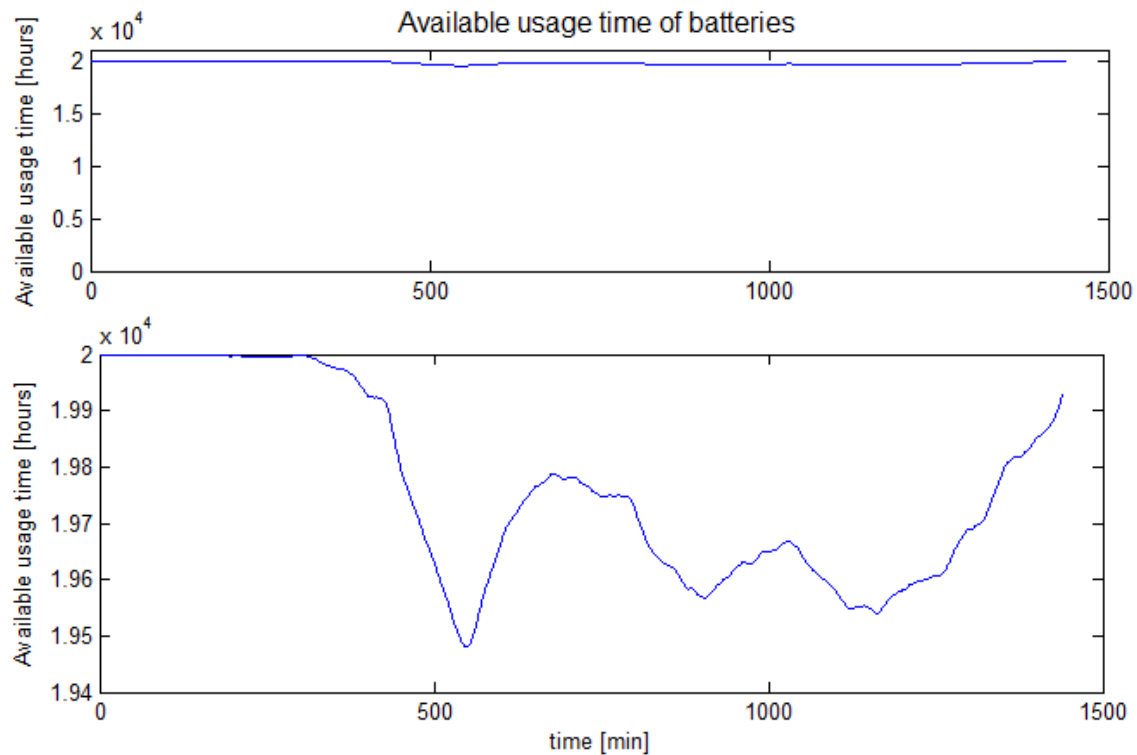


Figure 5.18. Available usage time for e-bikes

Costs analysis – KPIs

One of the highlights of the analysis of a transportation system is, of course, the cost analysis. Specifically, key performance indicators, or KPIs, are studied for each of the sections involved, which are defined below.

Agency cost:

It represents the total expense per hour that involves the operation of the bike-sharing system, that is to say, what it costs to the service administrator every hour. It is composed of the sum of the following factors:

Infrastructure costs: It is the cost of bicycles and stations. In this case, it is calculated from the unitary cost of each bike, which has already been joined by the proportional cost of the stations, according to the formula:

$$\text{Bike fleet} = \gamma \cdot m$$

Operation costs: It is the proportional cost of the maintenance of the service, according to the demand of the service. That is, bike maintenance, station maintenance and administrative costs. It is calculated as:

$$OC = \gamma_e \cdot \text{Avg. daily demand}$$

Repositioning costs: This is the cost involving reposition operations. Thus, it is the only dynamic cost to the operator, so that strategies to take on this are basic. It must be taken into account that the cost factor of each reposition team may vary for continuous and periodic reposition, especially if the last one is done at night, when the cost of operators is higher.

$$RC = C_t \cdot \frac{\text{Total time invested daily in reposition}}{24h}$$

User costs:

On the other hand, the cost perceived by the user in the system is given by the time it consumes to use it, as the access time, and specially the lost time due to bad service. Those costs are calculated as:

Access costs: It refers to the cost that has the user to access to the system, i.e., the time it takes to access it from its origin multiplied by the value of that time:

$$AC = \beta \cdot \frac{\text{cumulative daily access time}}{24h}$$

Bad service penalty: In the event that users have found empty or full stations at origin or destination and have had to divert its course, that time is counted as time lost, and so is perceived by them. That is why it is penalized by that time caused by bad service:

$$BSP = \beta_{tl} \cdot \frac{\text{cumulative daily lost access time}}{24h}$$

Death penalty: If, eventually, users do not find available bikes at origin they leave the system. As the service is especially designed to cover compulsory mobility trips, it is understood that the user would made it with other transportation mode. The cost of this trip would be the penalty of 'dying':

$$BSP = L \cdot \frac{\text{Total number of deaths}}{24h}$$

The costs factors proposed by the author and collected from previous researches (Llopis, A. 2015) are gathered in Table 5.4.

Parameter	Units	Value
γ	€/bike·h	0.05
γ_e	€/trip	0.62
C_t	€/team·h	13.03
β	€/h	11.4
β_{tl}	€/h	9.12
L	€	9.84

Table 5.4. Costs factors

Considering all this information, the costs resulting from the operation of the simulated system, which are shown in the following Table 5.5, are obtained.

Cost	€/h
Agency costs	
Infrastructure costs	300
Operation costs	1078.8
Continuous reposition costs	118.4
Periodic reposition costs	15.9
Total agency costs	1513.1
User costs	
Access costs	978.3
Bad service penalty	164.7
Death Penalty	986.5
Total user costs	2128.5
Total costs	3712.6

Table 5.5. Results KPI for the simulation with Barcelona's system parameters.

In addition, once calculated the total costs, other interesting facts can be obtained, such as the generalized cost per trip, which results from dividing the total cost by the average demand:

$$\text{Generalized cost per trip} = \frac{\text{Total costs [€/h]}}{\text{Avg. daily demand [trip/h]}}$$

Considering an average daily demand $\lambda = 1740$ trips/h, resulting from the simulation, this would give a cost of 2.13 €/trip.

Furthermore, the fare that should pay each user to cover the agency costs can be calculated, as:

$$\text{Fare} = \frac{\text{Total agency Costs [€/h]}}{\text{Avg. daily demand [trip/h]}}$$

Using the result obtained from the simulation is calculated a fare of 0.91 €/trip.

5.2. Optimum initial situation: an improved scenario

After obtaining a reference scenario for the situation of Barcelona's system, it is intended to evaluate a possible improvement in this system.

As it is seen in the previous section, one of the points that alter the most the final result is the balance point of reference, or initial state. This state determines all the simulation concerning the replacement, since repositioning teams try to bring the system to this balanced point, so that a correct choice of the initial occupation of the stations is central to the strategy of the operator.

With this purpose, it has been decided to use a reference point reached in other studies related to the repositioning (Jiménez, E. 2016), centered on similar routines to those used in this work, and although it is still being developed and evolved, it can give a substantial idea of the changes that the improvement of the equilibrium point may involve.

In the simulation of this scenario, therefore, any other input parameter has been changed, so the demand, the layout of the stations, reposition teams, decision parameters for users, etc. will remain the same, in order to compare the two results in a more direct way.

Thus, the occupation of the initial stations, which can be seen in Figure 5.19, compares this occupancy proposed to the reference occupancy for the first case simulated.

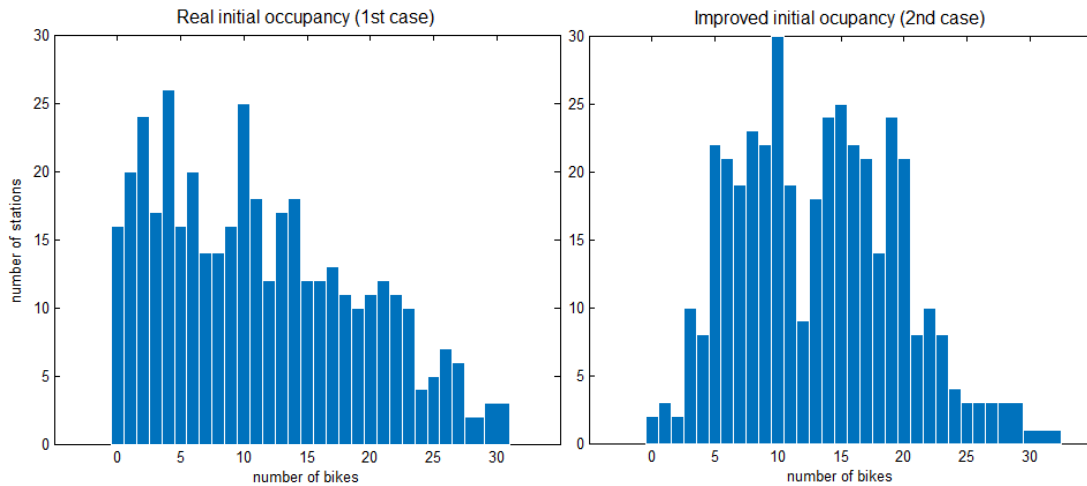


Figure 5.19. Comparison between real and improved initial occupancy of stations histograms

5.2.1. Visualization and analysis

Once simulated this new scenario, the attention must be directed to the occupancy of the stations and the accessibility that it provides to the system, as the proposed improvement will seek the goal of improving the availability of bicycles for users, preventing them from finding empty the station on origin, but also full stations at destination.

This way, it is possible to see how the percentage of stations simultaneously full and empty during the simulation has decreased considerably, as Figure 5.20 shows, especially for the percentage of empty.

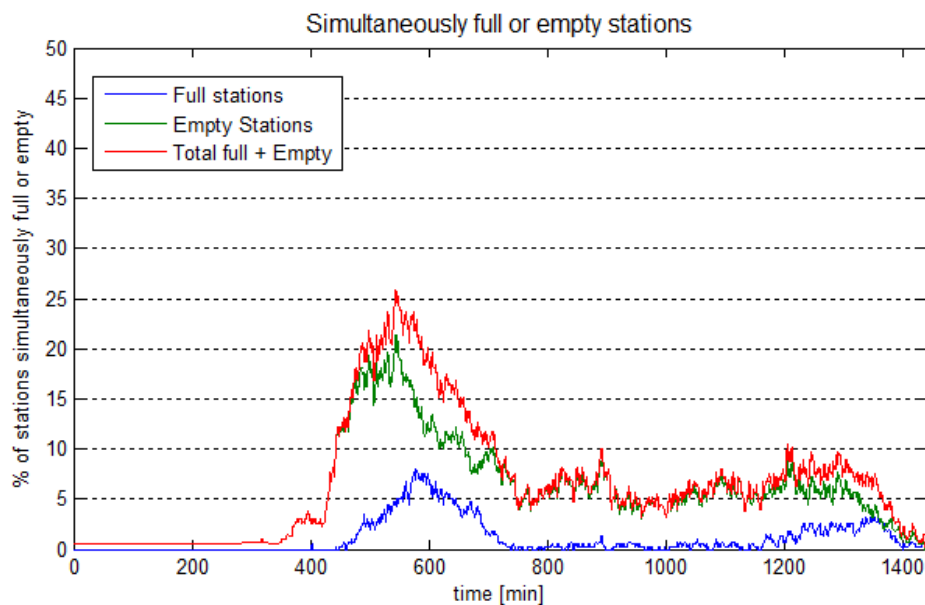


Figure 5.20. Simultaneously full or empty stations

Compared with the graph for the reference situation, in Figure 5.13, the fact that the initial peak of empty stations is maintained can be observed, which is explained because trucks must wait for alarms generated in stations where there is an excess of bicycles (avoiding journeys from station to station to pick up just a few bikes). This peak could be avoided by allowing trucks to load the stock remaining in the depot, so they could respond to empty stations alarms from the first moment.

In any case, once the peak hour demand ended, empty stations decrease significantly and stabilize around 5% of the total. In addition, the full stations are almost none, with minor variations.

The difference between empty and full stations remains virtually constant over the entire period simulated, which suggests that it would be appropriate to introduce even more bicycles in the system to the equilibrium point, but the target of increased accessibility has been accomplished. This can also be seen in the accumulated percentage of users who find empty stations, shown in Figure 5.21.

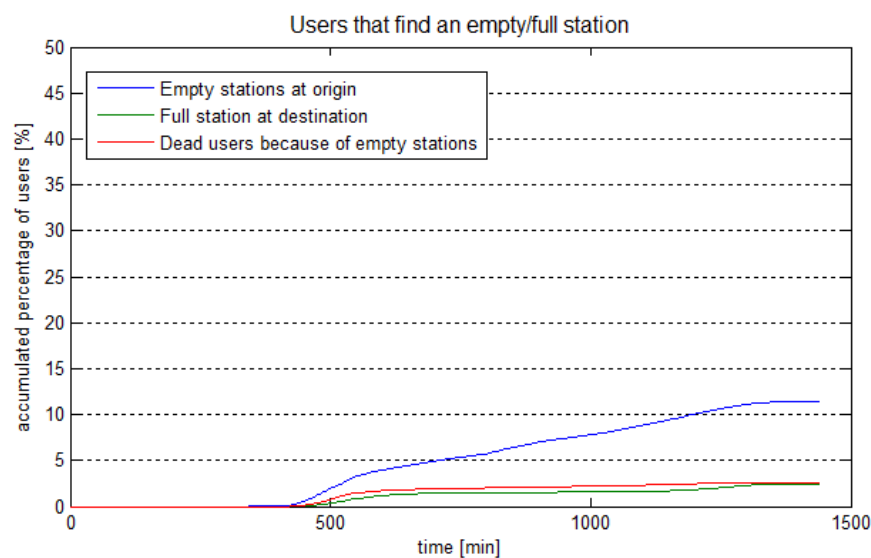


Figure 5.21. Accumulated percentage of users that find an empty or full station

The percentage of users who find an empty station, regarding to those who did it in the previous case and that was shown in Figure 5.8, has been reduced by more than 5 points, reaching an 11.5% of the total users. This result directly affects the number of users who leave the system for this reason, that also decrease to 2.6%, compared with 4.3% reached before, over half less.

In addition, better initial distribution, as had been discussed, optimizes truck trips, which in this case should visit fewer times the stations, and therefore will travel less distance, as shown by the results:

Total distance travelled = **1362 km**

Total time invested = **12843 minutes**

$$\text{Avg. distance} \frac{\text{travelled}}{\text{truck}} = \mathbf{104.8 \text{ km}}$$

$$\text{Avg. time} \frac{\text{invested}}{\text{truck}} = \mathbf{987.9 \text{ minutes}}$$

Despite the satisfactory results, the author of this master thesis wishes to recall that the routines proposed for reposition, as well as the whole reposition strategy, have been designed as mere approximation, and hopes that future studies will be conducted to optimize the rebalancing processes, for which this simulator can be helpful, as will be claimed in section 7.

6. CONCLUSIONS

A fully functional program for simulating bike-sharing systems from default layouts has been created, encompassing, for example, the layout of the stations in the city, their dimensions, the number of trucks destined for reposition tasks, etc.; and from basic performance parameters such as walking speed or maximum distance that users are willing to walk to access the system.

The program has been developed under open source license, which means that it is available to everyone who needs it without restrictions. The user therefore can read, modify, adapt or extend whatever it considers appropriate, which allows its development not to end where this master thesis does, but lays the foundation for further work that may improve in several respects, a few of which have been introduced in section 5.

The paradigm of object-oriented programming applied in the development allows the introduction of changes without performing only external changes. However, reposition functions, which are based on subroutines devised by the author of these lines, have been arranged through external functions that allow their relief at any time to improve the reposition study in bike sharing systems, basic strategic point for the management of these systems. The simulator, thus, is intended to be an attractive tool for anyone who carries out research in this field.

As confirmed, the above results show a precise and adjusted method operation, with close values to those observed in real implemented systems, as *Bicing* in Barcelona, and can serve as a valid approach for anyone who intends to study behaviors and patterns in these systems, as well as those who want to simulate virtual situations in systems not yet implemented. Obviously, the accuracy of the program will be or not enough, depending on the situation. It is up to the reader to decide on this aspect as well as to perform more tests on his or her own.

In conclusion, it can be determined that the objectives of the thesis have been met satisfactorily, achieving an effective, transparent and adaptable program.

7. FURTHER WORKS

As any software nowadays does, the simulator is in continuous development, and many improvements can be developed, about both technical details of the problem under study as practical issues, such as the interface user, data output system or efficiency of the code.

Some aspects that could be subject of study and short-term improvements referring to the simulator performance are suggested below:

- Metrics L1 vs. L2: The simulator is currently implemented in Euclidean metric (L2), but in many cases, and because of the morphology of cities, the L1 metric has better results. To choose between the two metrics would be a remarkable improvement.
- Incorporation of a destination and origin station considering the main direction of trips, preventing backtracking, which is unrealistic in short trips.
- Incorporation of the possibility that the user decides to wait when the station is full or empty and not going to another station, which is a pattern seen in the daily life of the system studied.

- Exploration of possible repositioning routines, because the current introduced strategies, as already mentioned, are contrived by the author of this thesis, without having been previously optimized and evaluated, although they seem to give a appropriate solution, but certainly improvable.
- Check the sections of code that can be inefficient, for improvements for reliability and speed.

Likewise, new lines of study are proposed. A wide field of study that directly affects mobility is the facilities that smartphones pose to users. This way, it could also be incorporated to the study the effect that might have informing by App to the users who are waiting in a station about a truck that is coming to carry out the repositioning task, or incorporate the possibility of booking a bike in origin and parking at destination in a better way.

Another interesting area to study, and in which the simulator can be a great support, is the business model of the agency and possible pricing schemes, such as annual fees, a fare per trip, consider temporary fractions, etc. Besides, possible rewards or bonus could be considered, for example, for taking bikes on full stations and leave them where needed, or taxes, i.e. booking bikes or parking and, in addition, the study of the effects that it would have on the demand.

It would also be possible to analyze the possibility of including a restriction to a single task complete by a truck in repositioning tasks, in order to avoid multiple visits. In view of the results obtained in the case studied, it seems that the improvement would not have any substantial jump, but may even be counterproductive, so it would be interesting to analyze it.

As for the case of study, explained in section 5, to improve its simulation results, it has to be noticed that there will be a change in the pattern of demand due to the electric bicycle. Currently, requests and returns parameters use the same array than traditional bikes, empirically collected from *Bicing*. This would imply a demand study, which could be carried out from data obtained from the current application of the electric model in Barcelona, or other electric bike-sharing systems applied directly with e-bikes, as Madrid's system.

Besides, it has been observed that the distribution of distances/travel times applied in the simulator does not conform entirely to reality, due to the application of a Poisson probability distribution. In any case, it would be more accurate by obtaining origin-destination data users from users, as well as quantized journey times, to obtain a distribution curve that fits the actual behavior of users to choose

the distance. Besides, outsourcing this distribution code in the simulator would be productive, to adapt it in each case simulated.

In the same direction, the equilibrium points for stations occupancy should be studied, depending on their demand. Applying the model of Barcelona, the initial situation registered on any given day has been used as equilibrium point, in this case on 7th May 2014 (Llopis, A. 2015), considering that this would be the situation after the complete repositioning operations, but without conducted studies of demand about the stations. The fact that the equilibrium point is not calibrated has led to certain errors accumulated over the entire repositioning simulation.

8. REFERENCES

- [1] Allan, R. J. (2010). Survey of Agent Based Modelling and Simulation Tools.
- [2] Barrios, J. A. (2011). Flexible Carsharing: Planning and Operation. University of California, Berkeley.
- [3] Bonabeau, E. (2002). Agent-based modeling: Methods and techniques for simulating human systems. Proceedings of the National Academy of Sciences of the United States of America.
- [4] Cassidy, M. J. and M. Mauch (2001). An observed traffic pattern in long freeway queues. Transportation Research Part A 35(2). 143-156.
- [5] DeMaio, P. (2009). Bike-sharing: History, Impacts, Models of Provision and Future. Public Journal of Transportation. Vol. 12. No 4. 2009. MetroBike LLC. Web. 15 March 2014.
- [6] EUNOIA – Evolutive User-Centric Networks for Intraurban Accessibility (2012). Urban models for transportation and spatial planning.
- [7] García-Palomares, J. et al (2012). Optimizing the location of stations in bike-sharing programs: A GIS approach. Applied Geography.

- [8] Gustafsson, L. and M. Sternad, (2010). "Consistent micro, macro, and state-based population modelling". *Mathematical Biosciences* 225 (2): 94–107.
- [9] ITDP -Institute for transportation and development policy (2013). *The Bike-share Planning Guide*.
- [10] Jiménez, E (2016). *Rebalancing strategies for bike-sharing systems: The case of "Bicing" Barcelona*. Universitat Politècnica de Catalunya.
- [11] Larsen, J. (2013). *Bike-Sharing Programs Hit the Streets in Over 500 Cities Worldwide*. Earth Policy Institute.
- [12] Larson, R.C. and A.R. Odoni (1981). *Urban Operations Research*.
- [13] Lin, J-R. and T-H. Yang (2011). Strategic design of public bicycle sharing systems with service level constraints. *Transportation Research Part E* 47, 284-294.
- [14] Llopis, A. (2015). *Modelling shared bicycle systems with application to e-bicycles*. Universitat Politècnica de Catalunya.
- [15] Martinez, L. M. et al (2012). An optimization algorithm to establish the location of stations of a mixed fleet biking system: an application to the city of Lisbon. *Procedia – Social and behavioral Sciences* 54.
- [16] McGarrity, S. (2009). *Introduction to object-Oriented programming in Matlab®*.
- [17] Morato, J. (2015). *Desarrollo en Matlab de software de cálculo de placas por MEF*. Universitat Politècnica de Catalunya.
- [18] Moura, J. L. et al (2012). A simulation-optimization approach to design efficient systems of bike-sharing. *Procedia – Social and behavioral Sciences* 54.
- [19] Munkres, J. (1957). "Algorithms for the Assignment and Transportation Problems". *Journal of the Society for Industrial and Applied Mathematics* Vol. 5, n. 1.
- [20] O'Kelly, M.E. (1986). The locating of interacting hub facilities. *Transportation Science* 20 (2), 92–106.
- [21] Shaheen, S, et al. (2015). *Public Bikeshaaring in North America During a Period of Rapid Expansion: Understanding Business Models, Industry Trends and User Impacts*. Mineta Transportation Institute (MTI). Retrieved 2014-11-05. pp. 5

- [22] Shaheen, S. and S. Guzman (2011). Worlwide bikesharing. Access Magazine No. 39. University of California Transportation Center.
- [23] Søren B. Jensen (2000), Free City Bike Schemes. City of Copenhagen, Conference Proceedings.
- [24] United States. U. S. Department of Transportation, Office of the Secretary of Transportation. "The Value of Travel Time Savings: Departmental Guidance for Conducting Economic Evaluations, Revision 2".
- [25] Vogel, P. (2014). Decision support for tactical resource allocation in bike sharing systems. Technische Unveristaet Braunschweig.
- [26] Bicing website.
[\[Online\] Available: https://www.bicing.cat/es/informacion/informacion-del-sistema](https://www.bicing.cat/es/informacion/informacion-del-sistema)
- [27] Wikipedia, "Agent-Based Model".
[\[Online\] Available: https://en.wikipedia.org/wiki/Agent-based_model](https://en.wikipedia.org/wiki/Agent-based_model)
- [28] Wikipedia, "Assignment Problem".
[\[Online\] Available: https://en.wikipedia.org/wiki/Assignment_problem](https://en.wikipedia.org/wiki/Assignment_problem)
- [29] Wikipedia, "Matlab".
[\[Online\] Available: http://es.wikipedia.org/wiki/MATLAB](http://es.wikipedia.org/wiki/MATLAB).
- [30] MathWorks, "Programación orientada a objetos en MATLAB".
[\[Online\] Available: http://es.mathworks.com/discovery/object-orientedprogramming.html](http://es.mathworks.com/discovery/object-orientedprogramming.html).
- [31] Wikipedia, "Voronoi diagram".
[\[Online\] Available: https://en.wikipedia.org/wiki/Voronoi_diagram](https://en.wikipedia.org/wiki/Voronoi_diagram)

APPENDIX A1: USER MANUAL

INTRODUCTION

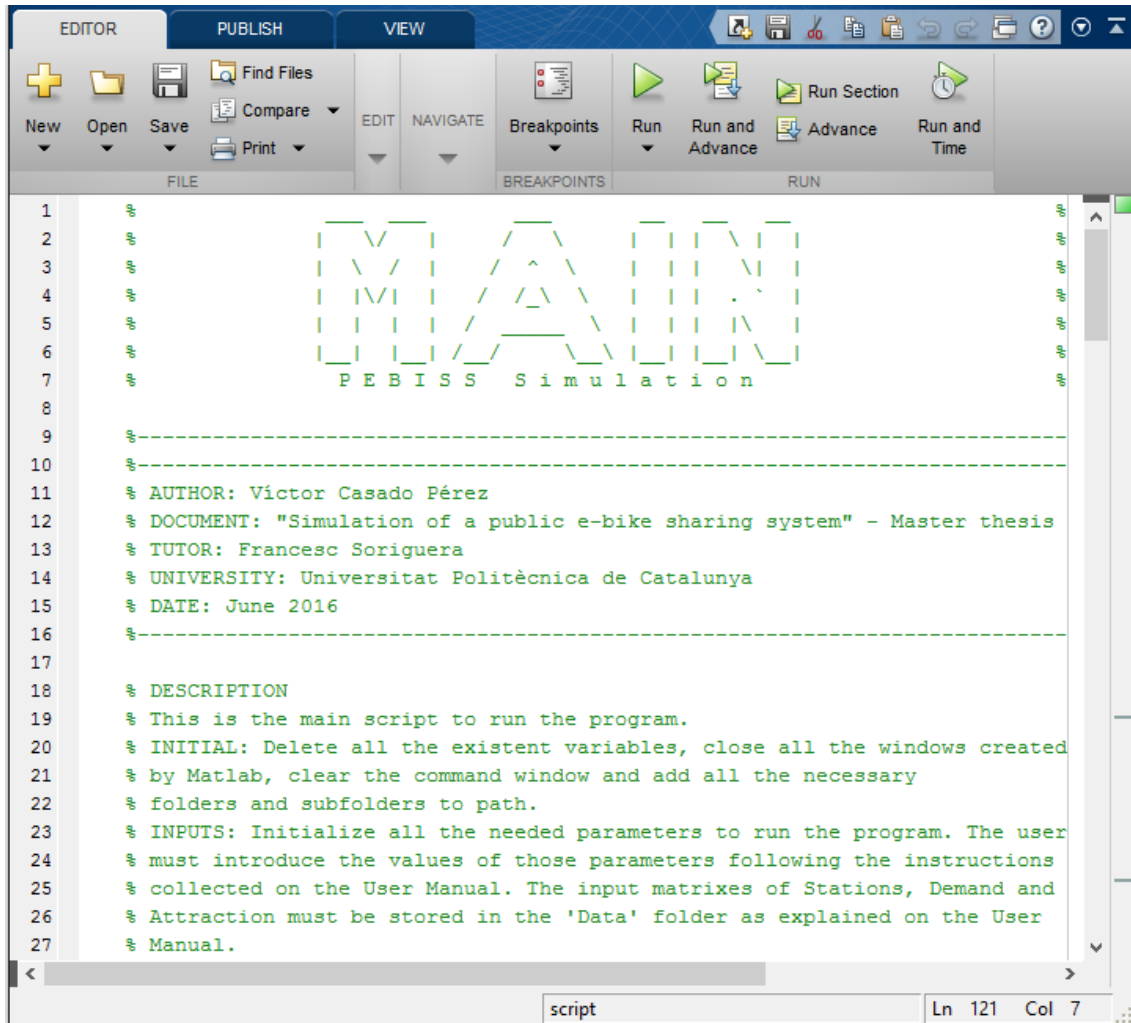
Welcome to the User manual of the Public E-Bike Sharing System Simulation.

This program simulates a public bike sharing system from series of basic parameters, which must be previously estimated. The software includes the possibility of the existence of a user's App that provides them knowledge about the conditions of the system in real time, basically the occupancy of the stations, so they can know if there are bikes available in nearby stations. Also includes the existence of reposition operations, either continuous or periodic, that allows analyzing efficiency and optimization of each of them. Besides, it consider the possibility of using electric bicycles (E-bikes), allowing also the comparison between them and the conventional bikes.

The full code has been written in Matlab®, between August and December 2015, as a part of a master thesis. For more information about the formulation of the problem, read the document "Simulation of a public e-bike sharing system", by Victor Casado.

MAIN SCRIPT

The main script, as the name suggests, is the most important file for the user. Named `MainSimulation.mat`, It contains a short description about the present software, and is the responsible to initialize the program. It is the only public file, as it calls the rest of functions, methods and classes to develop the simulation, so it should be the only script the user is acquainted with.



```
1 %  
2 %  
3 %  
4 %  
5 %  
6 %  
7 %  
8 %  
9 %-----  
10 %-----  
11 % AUTHOR: Víctor Casado Pérez  
12 % DOCUMENT: "Simulation of a public e-bike sharing system" - Master thesis  
13 % TUTOR: Francesc Soriguera  
14 % UNIVERSITY: Universitat Politècnica de Catalunya  
15 % DATE: June 2016  
16 %-----  
17 %  
18 % DESCRIPTION  
19 % This is the main script to run the program.  
20 % INITIAL: Delete all the existent variables, close all the windows created  
21 % by Matlab, clear the command window and add all the necessary  
22 % folders and subfolders to path.  
23 % INPUTS: Initialize all the needed parameters to run the program. The user  
24 % must introduce the values of those parameters following the instructions  
25 % collected on the User Manual. The input matrixes of Stations, Demand and  
26 % Attraction must be stored in the 'Data' folder as explained on the User  
27 % Manual.
```

Figure A1.1. MainSimulation.mat.

INPUTS

In this program the inputs are divided in two sections: The numerical simple parameters and the data matrixes. Both must be introduced into the program following the instructions specified below.

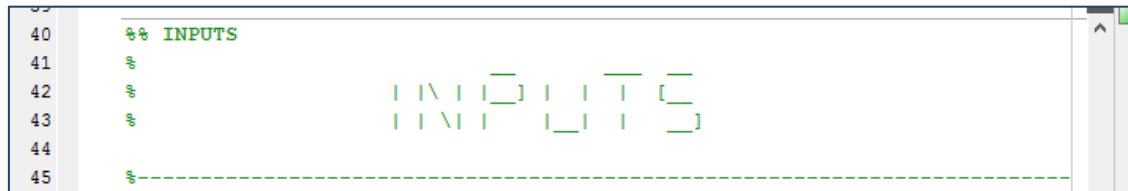


Figure A1.2. Inputs menu in MainSimulation.mat.

Simple parameters

The simple parameters are divided in five groups, depending on the object it refers (time, city, users, bikes or trucks). The user has to introduce the simple parameters on the `MainSimulation` script, only with a numerical value and in the correct units specified above them.

Here it is shown a relation between all the required parameters, a short description of them and their units:

Input parameters	Description	Units
Time		
TotalTime	Total time simulation time.	Min
TimeReDemand	Demand updating period.	Min
City		
Ebike	E-bike mode: it defines the kind of bikes used on the system: 0. Traditional bikes. 1. e-bikes.	-
App	App mode: it defines if it exists and App used by the users to know the current state of the system (stations occupancy): 0. No App service available. 1. App service available.	-
ContRepo	Continuous Reposition mode: it defines the kind of continuous reposition used in the system: 0. No continuous reposition. 1. Based on fixed routes (Peddling). 2. Based on alarms.	-

PeriodRepo	Periodic Reposition mode: it defines the kind of periodic reposition used in the system: 0. No periodic reposition. 1. Reposition at the end of the service.	-
m	Number of bikes in the system.	# of bikes
ContTrucks	Number of trucks in the system responsible for continuous reposition.	# of trucks
PeriodTrucks	Number of trucks in the system responsible for periodic reposition.	# of trucks
Per4Alarm	Lack of bikes or free spots percentage to set reposition alarm.	%
DepoPosition	A vector of position [x, y, z] in UTM data that locate the central depo for trucks.	m
Users		
Wmax	Maximum distance a user is willing to walk to access to the system.	m
WalkSpeed	Walking speed.	Km/h
AvgRideDist	Average riding distance.	m
BikeLoadUnload	Time needed to load/unload a bike on a station by a user.	Min
Bikes		
BikeSpeed	Riding speed.	Km/h
BatteryCharge	Time needed to charge an empty battery.	Min
BatteryConsume	Time needed to consume a full battery while uninterrupted riding.	Min
Trucks		
K	Capacity of the truck, number of bikes it can carry.	# of bikes
TruckSpeed	Truck's cruising speed.	Km/h
TimeLoadUnload	Time needed to load/unload a bike on a station by and operator of the system.	Min
expBikes	Expensive bikes: minimum movements needed to move a truck	-
Unitary costs		
Gam_b	Bike unitary cost	[€/bike·h]
Gam_e	Operations cost	[€/trip]
C_d	Reposition cost	[€/km]
Beta	Value of time	[€/h]
Beta_l	Extra value of lost time	[€/h]

Table A1.1. Input parameters needed for the simulation

Data matrixes

The program requires the existence of specific data matrixes to run. They must be saved in the folder '/Data' as individual .xls (excel) documents, and has to include the following information:

StationsData.xls: Contains the location, the capacity and the initial occupancy for each station (columns). Thus, the location is defined in the first 3 rows as X, Y, Z coordinates, i.e. in UTM coordinates; the fourth row defines the capacity of the station; and the fifth row defines the initial occupancy, that is to say the occupancy after reposition.

Boundary.xls: Contains the X, Y coordinates, i.e. in UTM coordinates, of a defined boundary for the service region, set in rows.

Demand.xls: Contains the demand of each station (columns) for each minute (rows), hence the matrix must have a number of rows equal to total time of simulation, specified with the simple parameter 'TotalTime'. However, the demand data must be in users/hour units.

Attraction.xls: It is like the Demand matrix but it contains the attraction rates of each station.

Before simulating, while creating the `City` object in which the entire simulation is based, its name of the can be defined in order to ease simulating different scenarios consecutively, for example, `MyCity`, `City1`, `City2`, etc.

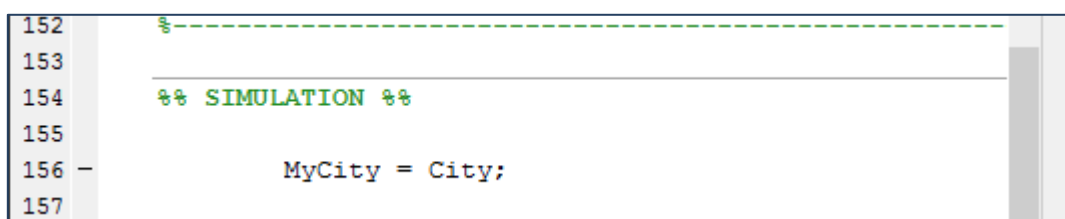
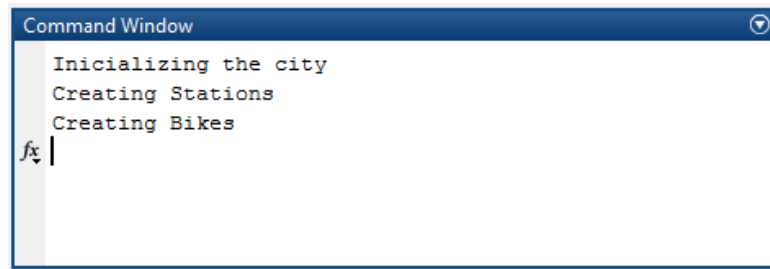


Figure A1.3. Creating the City object in MainSimulation.mat.

SIMULATION

Once all the inputs are introduced, the program can be run by pressing the 'Run' button from the script file, or calling the `MainSimulation` on the 'Command Window'. While the simulation is running the user will be able to notice some highlights on the command window to follow the course of the simulation.

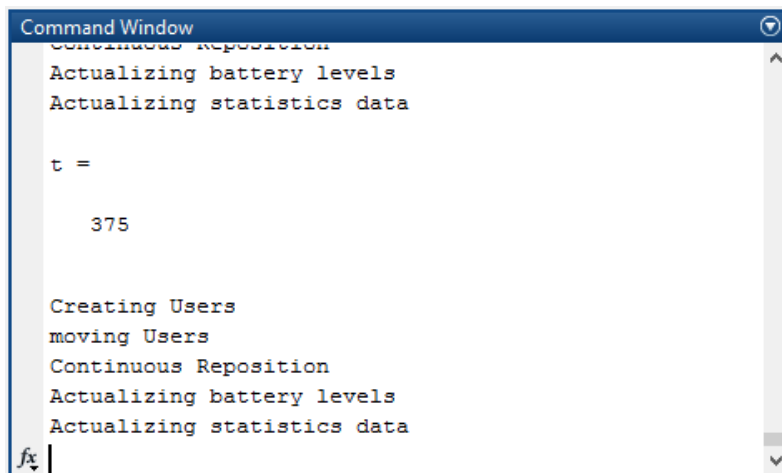
Firstly, it initializes the city, creating the stations, the bikes in the system and the trucks if any.



```
Command Window
Inicializing the city
Creating Stations
Creating Bikes
fx |
```

Figure A1.4. Command Window while initializing the City.

Then, the time simulation is started. For each time step the user of the simulator can read the current time in minutes, and a notification for every step while creating users (depending on the demand), moving the active users, moving the trucks if there is continuous reposition, and finally actualizing the collected data into the class objects of `Statistics`.



```
Command Window
Continuous Reposition
Actualizing battery levels
Actualizing statistics data

t =

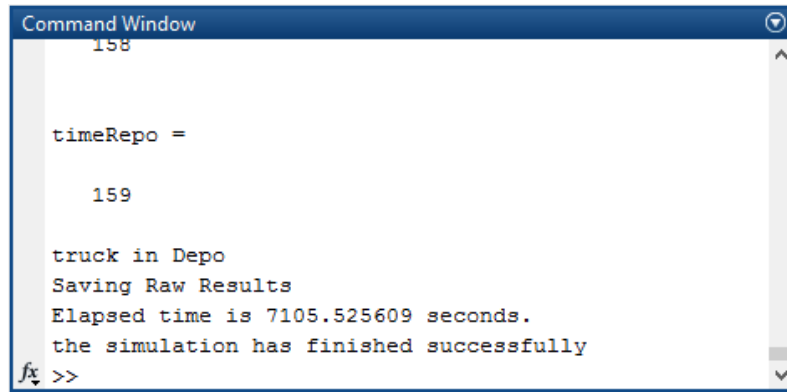
    375

Creating Users
moving Users
Continuous Reposition
Actualizing battery levels
Actualizing statistics data
fx |
```

Figure A1.5. Command Window while simulating

Finally, the global data will be filed in a `Statistics` object called `FinalData`, and if a periodic reposition is programed, it will start its simulation immediately. Their steps will be shown too on the command window.

The simulation ends when all the trucks for the periodic simulation are back in the Depot, and will be shown on the screen the message:



```
Command Window
158

timeRepo =

    159

truck in Depo
Saving Raw Results
Elapsed time is 7105.525609 seconds.
the simulation has finished successfully
fx >>
```

Figure: A1.6. Command Window at the end of the simulation

OUTPUTS

The main results are basically four matrixes stored in Results folder.

- An instant occupancy matrix, which collects the occupancy rate for each time step and each station.

File: Results/Instant_Occupancy.xls

- Two instant reposition operations matrix, one for continuous reposition and another for periodic reposition, which have all the reposition movements completed for each time step and each station as well.

Files: Results/Instant_ContinuousRepo_Operations.xls,
Results/Instant_PeriodicRepo_Operations.xls

- An accumulated origin-destination matrix that records all the trips done by the users.

File: Results/OD_Acumulated.xls

Additionally, the main object from `City` class will be stored in the same folder and all the data recorded on `Statistics` and `RepoStatistics` objects will be available at the end of the simulation, as properties of the `City` object called `vStatistics` or `vRepoStatistics` (containing every step information, for time simulation of the service and for reposition service), and `FinalData` or `FinalDataReport` (containing the final situation and some parameter averages reached during the simulation of the service and the time simulation of the reposition operations). These properties are specified in Appendix A2.

File: Results/Final_City_Variables.mat,

External reposition functions

In order to enable the proposals and checks of new methods and routines of reposition into the proposed simulator system, the functions that define the strategies to follow in this field have been outsourced to allow greater accessibility, avoiding having to enter directly into the code. For this, these functions are stored as separate scripts in the folder `Simulation/RepoAuxiliarFunctions/`.

In this line, the functions that are outsourced are described below:

Destination

It refers to the choice of destination for reposition trucks, i.e., the way they know which is the next station to visit during simulation. There are three files, one for each type of replacement supported by the simulator.

Files: `repoDest_Continuous1.mat`
 `repoDest_Continuous2.mat`
 `repoDest_Periodic.mat`

Function: `function LinksM = repoDest_mode(obj)`

Input: `obj` – City object. The current city simulated (i.e. `MyCity`).

Output: `LinksM` – two rows matrix. Links the trucks ID (1st row) with the stations ID (2nd row).

Operations

It refers to the calculation of number of operations to perform in a station, either take or leave bicycles.

File: `repoOperations_Num.mat`

Function: `function repoOperations_Num (vStations, obj)`

Input: `vStations` – Cell that stores all the station objects in a row.
 `obj` – Truck object. The current truck working.

To replace any of these functions, the file must be replaced by a file with the same name and with the same inputs and outputs (if needed) required by the current routine, described above.

Currently these functions are based on a assignment Pool system, described in detail in the document of the master thesis “Simulation of a public e-bike sharing system”, Victor Casado (2016). For more information, consult the document or revise the code.

LICENSE

The user of the program is free to use, redistribute, adapt, extend or modify the code without restriction. It requires, though, the recognition of the author and to keep the license in later iterations of the program or its parts.

APPENDIX 2: SUMMARY OF CLASS PROPERTIES AND METHODS

Class City

Properties	Description
Register of objects	
NumStations	Number of current stations in the system
vStations	Cell that stores all the station objects
vBikes	Cell that stores all the bikes objects
vUsers	Cell that stores all the users objects
vUsersAct	Cell that stores all the users objects that are active in the system on the current time step
vContTrucks	Cell that stores all the trucks objects responsible of continuous reposition.
vPeriodTrucks	Cell that stores all the trucks objects responsible of periodic reposition.
vRepoPool	Cell that stores the RepoPool object
vStatistics	Cell that stores all the Statistics objects
vContRepoStatistics	Cell that stores all the RepoStatistics objects responsible of continuous reposition.
vPeriodRepoStatistics	Cell that stores all the RepoStatistics objects responsible of periodic reposition.
FinalData	Specific Statistics object that contains data about the final situation and some averages.
FinalDataRepo	Specific RepoStatistics object that contains data about the final situation and some averages.
vAgencyCost	Vector that stores agency costs [infrastructure, operation, continuous repo, periodic repo, total]
vUserCost	Vector that stores user costs [access, bad service penalty, total]
vTotalCost	Vector that stores total costs [Total cost, generalized cost per trip, Fare]

User inputs	
boundM, DemandM, AttractM	Matrix of definition for the service region's boundary, and matrixes of demand and attraction for each station and minute, introduced as inputs.
TotalTime, TimeReDemand, m, Ebike, bikeSpeed, bikeLoadUnload, Wmax, AvgRideDist, App, ContRepo, PeriodRepo, ContTrucks, PeriodTrucks, DepoPosition, WalkSpeed, BatteryChargeRith, BatteryConsumerRith, K, TruckSpeed, TimeLoadUnload, expBikes, Per4Alarm, Gam_b, Gam_e, C_d, Beta, Beta_l, L	Several parameters introduced as inputs and stored as City properties to ease calling them, and to make them accessible at the end of the simulation, if required. Specified at Appendix A1: User manual.

Table A2.1. Properties of City class

Methods	Description
Constructor	
City	Creates the object and initialize the properties
Initializers of the city	
InitializeCity	Initialize all the needed parameters, objects and subroutines of the city and their initial values
setStations	Creates all the station defined in the StationData input matrix as a Patch object
createBikes	Creates all the Bikes objects
createTrucks	Creates all the Trucks objects, if there is reposition requested.
setRepoAreas	Assign to every station a zone ID that refers to a specific reposition truck. (Only for continuous reposition based on fixed routes).
routingTSP	Define routes following the travel salesman problem. (only for continuous reposition based on fixed routes).
Demand	
createUsers	Creates users at each station depending on the demand
moveUsers	Simulate the action of moving a user, either walking or riding a bike
actualizeUsersData	Actualize vUsersAct with the active users
actualizeUsersEnd	Actualize vUsers at the end of the simulation.

Reposition	
periodicRepo	Simulate the operations of periodic reposition.
continuousRepo	Simulate the operations of continuous reposition
definePeriodicRepoDestination	Call the external destination functions for periodic reposition
defineContinuousRepoDestination	Call the external destination functions for continuous reposition
defineTime2StaPeriod	Calculates the time to next station in periodic reposition
defineTime2StaCont	Calculates the time to next station in continuous reposition
defineTime2Depo	Calculates the time to central depo at the end of the periodic reposition
Data	
dataWhileSimulation	Collects relevant data of each time step into an object from Statistics class
finalData	Collects relevant data for the final situation of the system, and other important statistics about service time simulation
dataRepoSimulation	Collects relevant data of each time step of the reposition simulation into an object from RepoStatistics class
finalDataRepo	Collects relevant data for the final situation of the system once rebalanced, and other important statistics about reposition
rawOutputs	Saved the result matrixes into the Results folder
Time simulations	
timeSimulation	Calls methods on a structured way in a loop over time, implements the main time simulation
periodicRepoSimulation	Calls reposition methods on a structured way in a loop over time, implements the periodic reposition simulation in time

Table A2.2. Methods of City class

Class Patch

Properties	Description
ID	Station's Identification number
RepoAreaID	Identification number of the reposition area it belongs, if any. (Only for continuous reposition based on fixed routes).
Position	Vector of position coordinates x, y, z
Demand	Demand assigned to the influence area for a time lapse
Attract	Attraction assigned to the influence area for a time lapse
InfArea	Area of its influence zone
NumUsersInst	Number of users created in the influence area at the current time step
NumRequestInst	Number of requested bikes created on the station at the current time step
NumReturnsInst	Number of returned bikes created on the station at the current time step
NumRepoInst	Number of repositions done in the station at the current time step
Nplaces	Capacity of the station
Initialplaces	Occupancy of the station at beginning of the service (after reposition)
ONplaces	Occupancy of the station at the current time step
OFFplaces	Free parking spots in the station at the current time step
Alarm	Level of alarm for reposition: <ul style="list-style-type: none"> 0. Normal situation 1. Alarm: excess of bikes 2. Alarm: lack of bikes 3. Currently assigned to a truck
InfPoints	Boundary points of its influence zone
CrtGrid	Mesh of users creation

Table A2.3. Properties of Patch class

Methods	Description
Patch	Creates the object and initialize the properties
setPatch	Assign particular values to each property
setDemand	Assign the demand at the current time
setAttract	Assign the attraction at the current time
setInfArea	Generate a mesh of points where users are created
setPlacesOFF	Frees up a parking spot in the station
setPlacesON	Blocks a parking spot in the station
setAlarm	Assign the level of alarm
setStaticAlarm	Assign the level of alarm for periodic reposition

Table A2.4. Methods of Patch class

Class User

Properties	Description
ID	User's identification number
Origin	Vector of position x, y, z of the user's origin and station's ID assigned as origin
Destination	Vector of position x, y, z of the user's destination and station's ID assigned as destination
HistorialDestination	Collects all the IDs from the stations the user has had as destination
Position	Vector of current position x, y, z
BikeID	ID of the bike assigned or used by the user
WalkSpeed	Walking speed
Wmax	Maximum distance a user is willing to walk to access to the system
TimeCreation	Specific time of the simulation when the user is created
TimeOrig2Sta	Reminding time from the origin to the origin station
TimeUnload	Reminding time to take a bike from the station
TimeSta2Sta	Reminding time of riding trip
TimeSta2Dest	Reminding time from the destination station to real destination
DistRide	Distance that the user travelled by bike
DistRidePlus	Extra distance that the user travelled by bike due to full station at destination
TotalDistance	Total distance travelled by the user
TotalTimes	Collect the total time the user spend in the system, divided in sections (from origin to origin station, riding to de destination station, from de destination station to the real destination, looking for a parking spot to leave the bike if the origin station is full, extra time to destination due to leaving bike in a farther station and the total time in system)
TripCompleted	Defines the situation of the user in the system: <ul style="list-style-type: none"> 0. Still active 1. Trip completed 2. The user didn't find bikes at origin and left the system 3. The user didn't find e-bikes with enough battery at origin and left the system 4. The user left the system while looking for a destination 5. The user left the system due to lack of time to complete the trip before the service finish

Table A2.5. properties of Users class

Methods	Description
Constructor and settings	
User	Creates the object and initialize the properties
setUser	Assign particular values to each property
Origin definition	
defineOrigin	Define de real origin of the user in the influence area of the station where it has been created
defineNewOrigin	Define a new origin station if the first one is empty
Destination definition (Trip generation)	
defineDestination	Define a destination to the user and the closed station as destination station
defineNewDestination	Define a new destination station if the first one is full
ridingTime	Define the time needed to ride to the destination station.
Bikes assignation	
AssignBike	Assign a bike set in the station to the user
leaveBike	Leave the bike in the destination station

Table A2.6. Methods of Users class

Class Bike

Properties	Description
ID	Bike's Identification number
Position	Vector of current position x, y, z
BatteryLevel	Current level of the battery if e-bike
BatteryChargeRith	The percentage of battery that a connected e-bike charge in a minute
BatteryConsumeRith	The percentage of battery that an e-bike consume in a minute while riding
ONstation	Defines the situation of the bike: 0. Travelling 1. In station 2. On truck 3. In depot
StationID	ID of the assigned station
TotalDistance	Total distance that the bike have travelled

Table A2.7. Properties of Bike class

Methods	Description
Bike	Creates the object and initialize the properties
setBike	Assign particular values to each property
BatteryCharge	Simulate a battery charging while plugged, so every time step the battery level increases in the proportional part of charging rhythm
BatteryConsume	Simulate a battery consuming while riding, so every time step the battery level decrease in the proportional part of consuming rhythm

Table A2.8. Methods of Bike class

Class Truck

Properties	Description
General	
ID	Truck's Identification number
K	Vector of current position x, y, z
TruckSpeed	Truck's cruising speed.
TimeLoadUnload	Time needed to load/unload a bike on a station by and operator of the system.
expBikes	Expensive bikes: minimum movements needed to move a truck
HistStations	Record of visited stations
Position	Vector of current position x, y, z
BikesON	Number of bikes on the truck
vBikesON	Vector that collects the bike IDs of the bikes that it is carrying
Destination	Vector of position x, y, z and ID of the next station to rebalance
NumRepoActions	Number of operations to do on the next destination
ONstation	Defines the situation of the truck: 0. Travelling 1. In station 2. In depot
RepoEnded	Defines the situation of works: 0. Still repositioning 1. Reposition finished 2. In depot
Counters	
Time2Sta	Reminding time to next destination
TimeInSta	Reminding time in a station to complete its rebalancing
Time2Depo	Reminding time to depo if finished
TotalDistance	Total distance travelled during the operations
TotalActiveTime	Total time that the operator has been active
Only for continuous reposition based in fixed routes	
vOptRoute	Vector that contains the station IDs of the stations from its reposition area, in order of the route to follow
RouteDist	Distance of the route that the truck follows.
incBikes	Increment of bikes in its area of reposition

Table A2.9. Properties of Truck class

Methods	Description
Truck	Creates the object and initialize the properties
setTruck	Assign particular values to each property
repoInStation	Simulates the operation of reposition in a station
repoOperations	Define the number of operations to do in a station

Table A2.10. Methods of Truck class

Class RepoPool

Properties	Description
Trucks	Vector that stores the IDs of trucks that are looking for a destination
Stations	Vector that stores the IDs of the stations in alarm state
linksM	Matrix that link trucks (row 1) with stations (row 2)

Table A2.11. Properties of RepoPool class

Methods	Description
RepoPool	Creates the object and initialize the properties
clearPool	Clear the previous data recorded in the object
actPool	Actualize Trucks and Stations vectors
assignStations	Assign a station to each truck, following the Hungarian method (for continuous reposition).
assignStaticStation	Assign a station to each truck, following the Hungarian method (for periodic reposition).

Table A2.11. Methods of RepoPool class

Class Statistics

Properties	Description
Time	Time of data generation from the start of simulation
Users	
CreatedUsersNum	Number of users created
RealUsersNum	Number of users created that don't leave the system due to any restriction
CompletedNum	Number of trips completed
CompletedPer	Percentage of trips completed
DiedNum	Vector that collect the number of lefts from the system due to lack of bikes, slope and not finding destination
AvgDistRide	Average distance ride by bikes
DerivDestNum	Number of user that have changed their destination station due to full stations.
vDerivDist	Vector that stores the extra time of the derived users
AvgDerivDist	Average extra distance due to changing destination station
vAccessTime	Vector that collects access times of all users
AvgAccessTime	Average access time
Stations	
vOC	Vector That collects the stations occupation
AvgOC	Average stations occupation rate
EmptyStations	Number of empty stations
FullStations	Number of full stations
InstRequest	Number of requests (row 1) in specific stations (row 2)
InstReturns	Number of returns (row 1) in specific stations (row 2)
Bikes	
NumUsedBikes	Number of bikes in use
PerUsedBikes	Percentage of bikes in use
BikesOutPosition	Number of bikes out of their initial position
BatteriesLevel	Total time available on batteries
Raw simulation outputs	
OccM	Instant stations occupancy matrix
ODAcumM	Accumulated Origin-Destination Matrix
ODList	Chronologic Origin-Destination list

Table A2.12. Properties of Statistics class

Class RepoStatistics

Properties	Description
General	
Time	Time of data generation from the start of reposition
TotalTime	Time needed to complete the reposition
TotalRepoMov	Number of bikes relocated in reposition operations
RepoCost	Total cost of reposition
Stations	
RepoOperations	Number of reposition operations (row 1) in specific stations (row 2)
RpoVisits	Vector that stores instant visited stations IDs
Bikes	
BikesON	Current number of bikes on trucks
BikesOutPosition	Number of bikes out of initial position
Trucks	
TotalDistance	Total distance travelled by trucks
Raw simulation outputs	
RepoM	Instant reposition operations Matrix

Table A2.14. Properties of RepoStatistics class

